Radial Basis Functions Applied to Integral Interpolation, Piecewise Surface Reconstruction and Animation Control

---

A thesis submitted in partial fulfilment of the requirements for the

Degree

of Doctor of Philosophy in Mathematics

in the University of Canterbury

by Michael Keith Langton

University of Canterbury

2009

Abstract

This thesis describes theory and algorithms for use with Radial Basis Functions (RBFs), emphasising techniques motivated by three particular application areas.

In Part I, we apply RBFs to the problem of interpolating to integral data. While the potential of using RBFs for this purpose has been established in an abstract theoretical context, their use has been lacking an easy to check sufficient condition for finding appropriate parent basic functions, and explicit methods for deriving integral basic functions from them. We present both these components here, as well as explicit formulations for line segments in two dimensions and balls in three and five dimensions. We also apply these results to real-world track data.

In Part II, we apply Hermite and pointwise RBFs to the problem of surface reconstruction. RBFs are used for this purpose by representing the surface implicitly as the zero level set of a function in 3D space. We develop a multilevel piecewise technique based on scattered spherical subdomains, which requires the creation of algorithms for constructing sphere coverings with desirable properties and for blending smoothly between levels. The surface reconstruction method we develop scales very well to large datasets and is very amenable to parallelisation, while retaining global-approximation-like features such as hole filling. Our serial implementation can build an implicit surface representation which interpolates at over 42 million points in around 45 minutes.

In Part III, we apply RBFs to the problem of animation control in the area of motion synthesis—controlling an animated character whose motion is entirely the result of simulated physics. While the simulation is quite well understood, controlling the character by means of forces produced by virtual actuators or muscles remains a very difficult challenge. Here, we investigate the possibility of speeding up the optimisation process underlying most animation control methods by approximating the physics simulator with RBFs.

Acknowledgements

Firstly I would like to thank my supervisor Rick Beatson, who it has been a pleasure to work with, and who shares my enthusiasm for computers and making pretty pictures with them.

I would like to thank my office-mates over the years, Anton van Essen, Andrew Richens, Klaas Hartmann and Shannon Ezzat, for being good company and who together with all the other maths and stats postgrads (who I won't try to name, because I'll forget somebody) made the department a great place to be.

I would like to thank Steve Gourdie and Paul Brouwers for helping so willingly with my unusual computer requests.

I would like to gratefully acknowledge the financial support of the Tertiary Education Commission via a Top Achiever Doctoral Scholarship. It's a pity there will be no more of these, and I hope they'll come back in some form.

I would also like to thank my parents for setting such a good example and making flatting so easy.

Now, on with the maths!

Contents

Chapter 1

Overview

This thesis describes theory and algorithms for use with Radial Basis Functions (RBFs). Our emphasis is on techniques motivated by three application areas, integral interpolation, surface reconstruction and animation control. Working with these goals in mind, we have developed new theory, new analytical basic functions and new methods for deriving them; we have developed new algorithms for computation for these applications, and we have applied all these results to practical real-world data.

In Part I, we apply RBFs to the problem of interpolating to integral data. That is, constructing a function whose integrals over given compact regions such as lines or balls match given values. While the potential of using RBFs for this purpose has been established in an abstract theoretical context, their use has been lacking an easy to check sufficient condition for finding appropriate parent basic functions, and explicit methods for deriving integral basic functions from them. We present both these components here, the first through a Micchelli type theorem, as well as explicit formulations for line segments in any dimension and balls in three and five dimensions. We also apply these results to real-world track data.

In Part II, we apply Hermite and pointwise RBFs to the problem of surface reconstruction. In this application one begins with an unorganised set of surface points (e.g. from a laser scan) and constructs a representation of the underlying object shape, filling in the gaps between points with a smooth, continuous surface. RBFs are used for this purpose by representing the surface implicitly as the zero level set of a function in 3D space. As in the integral interpolation case, the theory of Hermite RBFs has been known for some time, but outside the field of differential equations there are hardly any practically oriented descriptions or applications in the literature. Hermite approximations are attractive for this application as they reduce the number of interpolation conditions and remove

the problem of specifying an off-surface distance, incorrect choice of which can cause problematic artefacts. In this work we develop a multilevel piecewise technique based on scattered spherical subdomains, which requires the creation of algorithms for constructing sphere coverings with desirable properties and for blending smoothly between levels. One contribution here is the development of a modified partition of unity method which is "level-aware" and automatically blends between approximation levels. The surface reconstruction method we develop scales very well to large datasets and is very amenable to parallelisation, while retaining global-approximation-like features such as hole filling. Scalability of the algorithm has been verified experimentally by an extensive series of numerical experiments, and compared against a competing method, where it performs very favourably. Our serial implementation can build an implicit surface representation which interpolates at over 42 million points in around 45 minutes.

In Part III, we apply RBFs to the problem of animation control in the area of motion synthesis. This is the problem of controlling an animated character whose motion is entirely the result of simulated physics. While the simulation is quite well understood and there exist several alternate methods for it, controlling the character by means of forces produced by virtual actuators or muscles remains a very difficult challenge. Here, we investigate the possibility of speeding up the optimisation process underlying most animation control methods by approximating the physics simulator with RBFs. This in itself is a difficult proposition, especially if we wish to maintain accuracy. We do not present concrete results in this section, but rather several mathematical steps towards them.

While the research is presented here divided cleanly by application, there are many threads common among the parts and there is considerable shared background, both in subject matter and in time, as repeatedly ideas arising from one application area sent us down new and promising avenues in another. RBFs are obviously the largest common factor, but Parts I and II share the theme of approximating data values that are not simply point evaluations, and the inherent complications arising from special basic functions that this requires, as well as the idea of fitting with greedy algorithms to approximate well with a minimum number of basis functions. Parts II and III share the themes of piecewise approximation and algorithms and data structures based on trees. Chronologically, the surface reconstruction of Part II was the final application area we investigated, and our success there is the result of our experience in researching integral interpolation and animation control.

Part I

Integral Interpolation

Chapter 2

Introduction

Part I of this thesis concerns interpolation problems in which the data to be interpolated consists of approximate averages of an unknown function over compact sets such as points, balls and line segments in $\mathbb{R}^n$. Such an *integral interpolation* approach is natural for datasets from many practical applications. In geostatistics and digital imaging, for example, data values may be area integrals. In computed tomography and ray tracing, data may be line integrals. Other sources such as mining or geographical surveying may give rise to *track data* consisting of point data sampled along lines. Here the data spacing is much denser in the direction of a track than between tracks, meaning it may be useful to group data points into line segments, each associated with a single combined scalar value.

In this work we focus specifically on straight line segments and balls, and adapt Radial Basis Function (RBF) techniques to interpolate to integrals over such sets. We will discuss the underlying mathematical theory, introduce explicit formulas making the techniques practical, and present numerical results of the method applied to real-world data. Large portions of this material have appeared in Beatson and Langton, 2007 [9].

2.1 Previous Work

Our integral interpolation approach can be viewed either as a generalisation of the one dimensional histospline technique of Boneva, Kendall and Stefanov [11] to scattered data (not on a grid) in two or three dimensions, or as a generalisation of Micchelli's [71] scattered point interpolation results to integral data.

Several generalisations of [11] have been given previously, and in [11] the authors do apply their grid-based technique in two dimensions using an approximation. As for analytical generalisations, Schoenberg [103] and de Boor [22] (an

appendix to [103]) discuss tensor product histosplines, Duchon [23, Theorems 2 and 4] has a general theory which covers integral interpolation by pseudo splines and polyharmonic splines, and Dyn and Wahba [25] present a theory that covers integral interpolation with polyharmonic splines.

Similarly, [71] has been generalised to work with integral data. Light [67] shows that the well known theorem of Micchelli [71] connecting complete monotonicity and pointwise conditional positive definiteness extends to integral conditional positive definiteness of order 0 and 1. The work of Iske [51] provides an alternative criterion for integral positive definiteness, relating it to the positivity of the Fourier or generalised Fourier transform.

Our contribution here covers several different *parent basic functions* and has an emphasis on the practical computational issues. In particular it emphasises the explicit formulas available for averages over line segments and balls which dramatically lower the number of floating point operations required to use the technique, making it practical for much larger problems.

2.2 Outline

In Chapter 3 we develop a Micchelli type theorem for integral positive definiteness, providing an easy to check sufficient condition for finding appropriate basic functions, and prove results useful for applications.

In Chapter 4 we derive explicit formulations for integral sources. In Section 4.1 we list several integrally strictly conditionally positive definite functions and derive closed form line segment sources from them. In Section 4.2 we describe the derivation of several ball sources in $\mathbb{R}^3$ and $\mathbb{R}^5$ (the resulting formulas are deferred until Appendix A).

Finally, in Chapter 5 we describe a greedy algorithm for fitting track data via integral interpolation, and compare this method to a point source approach.

Chapter 3

Theory

3.1 Integral Positive Definiteness

We will derive various integral sources (basic functions associated with a compact non-empty set rather than a point) from parent basic functions Φ which are strictly integrally conditionally positive definite in the sense defined below. These definitions recall one of Cheney and Light [19, page 133].

Definition 1. *A continuous real-valued kernel* $\Phi(\cdot,\cdot)$ *will be called* integrally positive definite *on* $\mathbb{R}^n$ *if*

1. $\Phi(\boldsymbol{x},\boldsymbol{y}) = \Phi(\boldsymbol{y},\boldsymbol{x})$ *for all* $\boldsymbol{x}$, $\boldsymbol{y}$ *in* $\mathbb{R}^n$.

2. *For every compactly-supported regular Borel (signed) measure* μ,

$$\iint \Phi(\boldsymbol{x},\boldsymbol{y})d\mu(\boldsymbol{x})d\mu(\boldsymbol{y}) \geq 0.$$

The kernel Φ *will be called* integrally strictly positive definite *on* $\mathbb{R}^n$, *denoted* $\mathsf{ISPD}(\mathbb{R}^n)$, *if the inequality is strict whenever* μ *is nonzero.*

Definition 2. *A continuous real-valued kernel* $\Phi(\cdot,\cdot)$ *will be called* integrally conditionally positive definite *of order* k *on* $\mathbb{R}^n$ *if*

1. $\Phi(\boldsymbol{x},\boldsymbol{y}) = \Phi(\boldsymbol{y},\boldsymbol{x})$ *for all* $\boldsymbol{x}$, $\boldsymbol{y}$ *in* $\mathbb{R}^n$.

2. *For every compactly-supported regular Borel (signed) measure* μ *on* $\mathbb{R}^n$ *such that* $\int_{\mathbb{R}^n} q(\boldsymbol{x})d\mu(\boldsymbol{x}) = 0$ *for all* $q \in \pi^n_{k-1}$,

$$\iint \Phi(\boldsymbol{x},\boldsymbol{y})d\mu(\boldsymbol{x})d\mu(\boldsymbol{y}) \geq 0,$$

where π_{k-1}^n *denotes the space of polynomials of total degree at most* $k-1$ *in* n *variables. The kernel* Φ *will be called* integrally strictly conditionally positive definite *of order* k *on* $\mathbb{R}^n$*, denoted* $\mathsf{ISPD}_k(\mathbb{R}^n)$*, if the inequality is strict whenever* μ *is nonzero.*

A measure μ is called *compactly supported* when the value of $\mu(f)$ depends only on the values of f within a compact set. Several examples of $\mathsf{ISPD}_k(\mathbb{R}^n)$ basic functions are listed in sections 4.1 and 4.2 below.

The definition above is a generalisation of the well known definition of pointwise strict conditional positive definiteness which arises when ordinary pointwise, or Lagrange, interpolation is considered. The ordinary pointwise definition will be recovered if we restrict μ to be a finite weighted sum of point evaluations. That is, if we require

$$\mu = \sum_{j=1}^{m} c_j \delta_{\boldsymbol{x}_j},$$

so that

$$\mu(q) = \sum_{j=1}^{m} c_j q(\boldsymbol{x}_j) \text{ and } \iint \Phi(\boldsymbol{x},\boldsymbol{y})\, d\mu(\boldsymbol{x}) d\mu(\boldsymbol{y}) = \sum_{i=1}^{m}\sum_{j=1}^{m} c_i c_j \Phi(\boldsymbol{x}_i,\boldsymbol{x}_j).$$

The motivation behind the current definition is that if $D \subset \mathbb{R}^n$ is compact then the dual $C(D)^*$ of $C(D)$ (the set of continuous functions from D to $\mathbb{R}$) is the set of continuous linear functionals $\mu(f) = \int_D f(\boldsymbol{x})\, d\mu(\boldsymbol{x})$, with μ a regular Borel measure on D. (This is a form of the Riesz Representation Theorem; see for example Rudin [97].) Hence, if we want a definition of positive definiteness appropriate for interpolation problems which involve a mixture of point values and weighted averages it is natural to require only continuity for Φ and to allow functionals that are regular Borel measures. If we were concerned with Hermite interpolation then a different definition of positive definite, requiring at least greater smoothness, would be appropriate. See Wu [133], Sun [110], and Narcowich [77] for some possibilities.

Given a function f, and m compactly supported regular Borel measures μ_i, we will seek an interpolant s such that

$$\mu_i(s) = \mu_i(f), \quad \text{for all } 1 \le i \le m.$$

Often we will not know f but only some observations of it. For example if $\mu_i(f)$ is an average over a ball $\mathcal{B}$ and $f_1, \ldots, f_N$ are observations of $f(\boldsymbol{x})$ at points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ then

$$\mu_i(f) = \int_{\mathcal{B}} f(\boldsymbol{x}) d\mu_i(\boldsymbol{x}) = \text{average value of } f \text{ on } \mathcal{B} \approx \frac{1}{\#\{j : \boldsymbol{x}_j \in \mathcal{B}\}} \sum_{j:\boldsymbol{x}_j \in \mathcal{B}} f_j.$$

Hence it is reasonable to take the experimentally observed average value as an approximation to the unknown continuous average, and interpolate to it.

We will need the following definition.

Definition 3. *A set of linear functionals μ_i, $1 \le i \le m$, will be called* unisolvent *for π^n_{k-1} if*

$$q \in \pi^n_{k-1} \text{ and } \mu_j(q) = 0 \text{ for all } 1 \le j \le m \implies q \text{ is the zero polynomial.}$$

We consider integral interpolation problems of the following form:

Problem 1 (Integral interpolation)**.** *Let Φ be an $\mathsf{ISPD}_k(\mathbb{R}^n)$ kernel, and let $\mu_1, \ldots, \mu_m$ be linearly independent compactly supported linear functionals on $C(\mathbb{R}^n)$ which are unisolvent for π^n_{k-1}. Let $b_1, \ldots, b_m$ be m real values. Find a function s of the form*

$$s(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{j=1}^{m} c_j \int_{\mathbb{R}^n} \Phi(\boldsymbol{x}, \boldsymbol{y})\, d\mu_j(\boldsymbol{y}), \quad p \in \pi^n_{k-1}, \tag{3.1}$$

such that

$$\int s(\boldsymbol{x})\, d\mu_i(\boldsymbol{x}) = b_i, \quad 1 \le i \le m,$$

and

$$\sum_{j=1}^{m} c_j \int q(\boldsymbol{x})\, d\mu_j(\boldsymbol{x}) = 0, \quad \text{for all } q \in \pi^n_{k-1}.$$

In the pointwise interpolation case the function s has the form

$$s(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{j=1}^{m} c_j \Phi(\boldsymbol{x}, \boldsymbol{x}_j). \tag{3.2}$$

The expression (3.1) for s justifies the name *parent basic function* for Φ used previously, since when interpolating with general functionals, we seek an approximation made up of polynomials plus functions like $\mu_j(\Phi(\boldsymbol{x}, \cdot)) = \int_{\mathbb{R}^n} \Phi(\boldsymbol{x}, \boldsymbol{y})\, d\mu_j(\boldsymbol{y})$ derived from Φ. Under weak conditions on the geometry (independence) of the functionals the derived functions form a *compatible family*. That is, they form a family of functions for which the corresponding interpolation matrix has positive definiteness properties making the interpolation problem uniquely solvable.

In order to be more concrete let $\ell = \dim\left(\pi^n_{k-1}\right)$ and $\{p_1, \ldots, p_\ell\}$ be a basis of π^n_{k-1}. Then the integral interpolation problem above can be rewritten in matrix form as:

Problem 2 (Integral interpolation matrix form)**.** *Solve*

$$\begin{bmatrix} G & P \\ P^T & O \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \tag{3.3}$$

for vectors $\boldsymbol{c}$ *and* $\boldsymbol{a}$ *where* G *is* $m \times m$ *with*

$$G_{ij} = \iint \Phi(\boldsymbol{x}, \boldsymbol{y})\, d\mu_i(\boldsymbol{x}) d\mu_j(\boldsymbol{y}),$$

P *is* $m \times \ell$ *with* $P_{ij} = \int_{\mathbb{R}^n} p_j(\boldsymbol{x})\, d\mu_i(\boldsymbol{x})$*, and* $p = \sum_{j=1}^{\ell} a_j p_j$*.*

Considering this problem, arguments similar to those used in the pointwise positive definite case show:

Theorem 1. *Let* Φ *be an* $\mathsf{ISPD}_k(\mathbb{R}^n)$ *kernel. Let* $\mu_1, \ldots \mu_m$ *be independent compactly supported regular Borel measures on* $C(\mathbb{R}^n)$ *which are unisolvent for* π^n_{k-1}*. Then the integral interpolation problem, Problem 1, has a unique solution. The coefficients of this solution may be found by solving the linear system of Problem 2.*

A proof of this theorem is given in Section 3.2.

The theory above is a direct generalisation of the pointwise, or Lagrange, interpolation case and is very satisfactory. However, integral interpolation would be impractical for large problems if numerical quadrature was required in order to evaluate the fitted function s of equation (3.1), and if two dimensional or higher quadrature had to be used to form the entries of the matrix G of the fitting problem, Problem 2. Fortunately, usually for averages over line segments no quadrature is needed to evaluate the interpolant s, and only univariate quadrature is needed in finding the entries of the matrix G. For averages over balls usually all needed quantities can be given in closed form, and no quadrature is needed in either evaluation or fitting.

The layout of this chapter is as follows. In Section 3.2 we recall and enhance some results of Light [67]. These provide us with a rich collection of integrally strictly conditionally positive definite functions. In Section 3.3 we discuss a sufficient condition for unisolvency and a way of replacing the linear system (3.3) with a positive definite system.

In the rest of the chapter we will assume that the Φ is of the special form $\Phi(\boldsymbol{x}, \boldsymbol{y}) = \psi(\|\boldsymbol{x} - \boldsymbol{y}\|_2)$ for some $\psi\colon \mathbb{R} \mapsto \mathbb{R}$. We will therefore change notation and write $\Phi(\boldsymbol{x})$ where Φ is radial. This amounts to replacing $\Phi(\boldsymbol{x}, \boldsymbol{y})$ by $\Phi(\boldsymbol{x} - \boldsymbol{y})$ in everything above.

3.2 Proofs

In this section we discuss integral interpolation and interpolation with general functionals. We discuss an analogue due to Light [67] of Micchelli's Theorem for completely monotone functions. This provides a very easy to check sufficient condition for integral strict conditional positive definiteness of order k in every dimension.

Consider Hermite piecewise cubic interpolation in one variable with data at the points $t_0 < t_1 < \ldots < t_m$. After some work it is possible to express such an interpolant in the form

$$h(x) = p_1(x) + \sum_{i=0}^{m} c_i |x - t_i|^3 - \sum_{i=0}^{m} d_i 3(x - t_i)|x - t_i|$$

where

$$\sum_{i=0}^{m} c_i = 0 = \sum_{i=0}^{m} (d_i + c_i t_i) .$$

In this expression note that the derivative interpolations we wish to make at the points t_i have introduced kernels $\frac{d}{dy}\Phi(x-y)$ into the spline/radial basis function. Here $\Phi(x-y) = |x-y|^3$ is the usual kernel arising when natural cubic spline interpolation is viewed as an example of Radial Basis Function interpolation.

The example above is one instance of a much more general pattern. Specifically that when interpolating with general functionals μ_i in a symmetric way, the kernels $\Phi(x-x_i)$ appropriate for point evaluations should be replaced by kernels $\mu_i(\Phi(x-\cdot))$. The pattern is clear in the papers of Iske [51], Narcowich [77], Franke and Schaback [30], and others. It is this pattern which motivated us to set up the integral interpolation problem as in Problem 1.

In order to use the solution to Problem 1 given in Theorem 1 we need to show that there exist some radial functions Φ which are $\mathsf{ISPD}_k(\mathbb{R}^n)$. Note that it is easy to show (e.g. Wendland [129, page 67]) that strict pointwise positive definiteness of Φ implies integral positive definiteness of Φ. Unfortunately, this is not enough, the strictness is essential for the poisedness of the integral interpolation problem. Narcowich [77] notes that a function's (pointwise) strict positive definiteness does not imply that it is integrally strictly conditionally positive definite, and points out a counter-example by Ron and Sun [96].

To identify some $\mathsf{ISPD}_k(\mathbb{R}^n)$ functions, one can modify A.L. Brown's elegant density proof in [13], or otherwise show:

Lemma 1 (A.L. Brown)**.** *Let $\sigma > 0$. The Gaussian $\Phi(\boldsymbol{x}) = \exp(-\sigma\|\boldsymbol{x}\|^2)$ is integrally strictly positive definite on $\mathbb{R}^n$ for every n.*

Then one can generalise the result of Micchelli [71] for the pointwise positive

definite case obtaining the following result.

Theorem 2 (W.A. Light [67]). *Let $\eta \in C[0,\infty)$, with $(-1)^k\eta^{(k)}$ completely monotonic and not constant on $(0,\infty)$. Then $\Phi(\boldsymbol{x}) = \eta(\|\boldsymbol{x}\|^2)$ is integrally strictly conditionally positive definite of order k on $\mathbb{R}^n$, for all n.*

This theorem provides us with a plentiful collection of integrally strictly conditionally positive definite functions. See Section 4.1 for some examples. Light actually proved the Theorem for the cases $k = 0$ and $k = 1$. We give a proof along the lines of Micchelli [71] for general k below, but first we need an analogue of [71, Lemma 3.1]:

Lemma 2. *Let μ be a compactly supported regular Borel measure such that $\int_{\mathbb{R}^n} q(\boldsymbol{x})\, d\mu(\boldsymbol{x}) = 0$ for all $q \in \pi^n_{k-1}$. Then*

$$(-1)^k \iint \|\boldsymbol{x}-\boldsymbol{y}\|^{2k}\, d\mu(\boldsymbol{x})\, d\mu(\boldsymbol{y}) \geq 0, \tag{3.4}$$

and equality holds in (3.4) *if and only if*

$$\int q(\boldsymbol{x})\, d\mu(\boldsymbol{x}) = 0, \text{ for all } q \in \pi^n_k. \tag{3.5}$$

Proof. We use almost identical reasoning to Micchelli [71], beginning by expanding the norm term inside the integral:

$$\begin{aligned}
\|\boldsymbol{x}-\boldsymbol{y}\|^{2k} &= ((\boldsymbol{x}-\boldsymbol{y})\cdot(\boldsymbol{x}-\boldsymbol{y}))^k && (3.6)\\
&= (\boldsymbol{x}\cdot\boldsymbol{x} - 2\boldsymbol{x}\cdot\boldsymbol{y} + \boldsymbol{y}\cdot\boldsymbol{y})^k && (3.7)\\
&= \left(\left(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2\right) - 2\boldsymbol{x}\cdot\boldsymbol{y}\right)^k && (3.8)\\
&= \sum_{l=0}^{k} \binom{k}{l} \left(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2\right)^{k-l} (-2\boldsymbol{x}\cdot\boldsymbol{y})^l && (3.9)\\
&= \sum_{l=0}^{k} (-1)^l 2^l \binom{k}{l} \sum_{h=0}^{k-l} \binom{k-l}{h} \|\boldsymbol{x}\|^{2h} \|\boldsymbol{y}\|^{2(k-l-h)} (\boldsymbol{x}\cdot\boldsymbol{y})^l . && (3.10)
\end{aligned}$$

(Note that there is a typographical error in [71], the $k-l$ of the second choose function being written as k. This is carried through the rest of the proof there, but does not affect the outcome.)

These summands are polynomials in $\boldsymbol{x}$ and $\boldsymbol{y}$, and the condition on μ means that $\iint p(\boldsymbol{x})q(\boldsymbol{y})\, d\mu(\boldsymbol{x})\, d\mu(\boldsymbol{y})$ will be zero whenever either p or q is of degree $< k$. We have a polynomial in $\boldsymbol{x}$ of degree $< k$ if

$$2h + l < k,$$

and a polynomial in $\boldsymbol{y}$ of degree $< k$ if

$$2(k-l-h)+l<k \qquad \Leftrightarrow \qquad 2k-l-2h<k \qquad \Leftrightarrow \qquad k<2h+l.$$

I.e., for non-zero summands, $2h+l=k$. So $k-l$ is even, and $h=\frac{k-l}{2}$. In the next part of the proof we need to use a multi-index $\boldsymbol{\alpha}=(\alpha_1,\alpha_2,\ldots,\alpha_n)$, where $\boldsymbol{x}^{\boldsymbol{\alpha}}=x_1^{\alpha_1}x_2^{\alpha_2}\ldots x_n^{\alpha_n}$, $|\boldsymbol{\alpha}|=\sum_{i=1}^n\alpha_i$, and the multinomial $\binom{l}{\boldsymbol{\alpha}}$ is defined as

$$\binom{l}{\boldsymbol{\alpha}}=\binom{l}{\alpha_1,\alpha_2,\ldots,\alpha_n}=\frac{l!}{\alpha_1!\alpha_2!\ldots\alpha_n!}.$$

We continue the proof, starting with part of the left hand side of Equation 3.4.

$$\begin{aligned}
&\iint \|\boldsymbol{x}-\boldsymbol{y}\|^{2k}\,d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=\iint \sum_{\substack{l=0\\ k-l \text{ even}}}^{k}(-1)^l 2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\|\boldsymbol{x}\|^{k-l}\|\boldsymbol{y}\|^{k-l}\,(\boldsymbol{x}\cdot\boldsymbol{y})^l\;d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=\iint(-1)^k\sum_{\substack{l=0\\ k-l \text{ even}}}^{k}2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\|\boldsymbol{x}\|^{k-l}\|\boldsymbol{y}\|^{k-l}\sum_{|\boldsymbol{\alpha}|=l}\tbinom{l}{\boldsymbol{\alpha}}\boldsymbol{x}^{\boldsymbol{\alpha}}\boldsymbol{y}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=(-1)^k\sum_{\substack{l=0\\ k-l \text{ even}}}^{k}2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\iint\|\boldsymbol{x}\|^{k-l}\|\boldsymbol{y}\|^{k-l}\sum_{|\boldsymbol{\alpha}|=l}\tbinom{l}{\boldsymbol{\alpha}}\boldsymbol{x}^{\boldsymbol{\alpha}}\boldsymbol{y}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=(-1)^k\sum_{\substack{l=0\\ k-l \text{ even}}}^{k}2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\sum_{|\boldsymbol{\alpha}|=l}\tbinom{l}{\boldsymbol{\alpha}}\iint\|\boldsymbol{x}\|^{k-l}\|\boldsymbol{y}\|^{k-l}\boldsymbol{x}^{\boldsymbol{\alpha}}\boldsymbol{y}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=(-1)^k\sum_{\substack{l=0\\ k-l \text{ even}}}^{k}2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\sum_{|\boldsymbol{\alpha}|=l}\tbinom{l}{\boldsymbol{\alpha}}\int\|\boldsymbol{y}\|^{k-l}\boldsymbol{y}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{y})\int\|\boldsymbol{x}\|^{k-l}\boldsymbol{x}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{x}),
\end{aligned}$$

where we have applied Fubini's theorem. Thus,

$$\begin{aligned}
&(-1)^k\iint\|\boldsymbol{x}-\boldsymbol{y}\|^{2k}\,d\mu(\boldsymbol{x})d\mu(\boldsymbol{y})\\
&=\sum_{\substack{l=0\\ k-l \text{ even}}}^{k}2^l\tbinom{k}{l}\tbinom{k-l}{(k-l)/2}\sum_{|\boldsymbol{\alpha}|=l}\tbinom{l}{\boldsymbol{\alpha}}\left(\int\|\boldsymbol{x}\|^{k-l}\boldsymbol{x}^{\boldsymbol{\alpha}}\,d\mu(\boldsymbol{x})\right)^2 \qquad (3.11)\\
&\geq 0. \qquad (3.12)
\end{aligned}$$

This proves (3.4). If (3.5) holds, each integral in (3.11) is zero since $\|\boldsymbol{x}\|^{k-l}\boldsymbol{x}^{\boldsymbol{\alpha}}$ is a polynomial of degree $\leq k$ (as $|\boldsymbol{\alpha}|=l$ and $k-l$ is even), and so each summand is annihilated. Conversely, if (3.12) is zero, all the integral terms in (3.11) must

be zero. Then in particular, when $l = k$ we have

$$\int \boldsymbol{x}^{\boldsymbol{\alpha}}\, d\mu(\boldsymbol{x}) = 0, \quad \text{for all } |\boldsymbol{\alpha}| = k.$$

Therefore

$$\int q(\boldsymbol{x})\, d\mu(\boldsymbol{x}) = 0$$

for all homogeneous polynomials q of degree k. Since μ already annihilates π^n_{k-1}, (3.5) follows. □

Proof of Theorem 2. Consider a function $\eta \in C[0,\infty)$ for which $(-1)^k\eta^{(k)}(t)$ is completely monotone but nonconstant on $(0,\infty)$. Then $(-1)^k\eta^{(k)}(t)$, being bounded below by 0 and non-increasing, necessarily tends to a finite nonnegative limit, c, as $t \to \infty$. Using the Bernstein-Widder theorem there is a finite nonnegative Borel measure ν so that

$$(-1)^k\eta^{(k)}(t) = \int_0^\infty e^{-t\sigma}\, d\nu(\sigma),$$

for all $t > 0$. As noted in [19, page 135] $c = \lim_{t\to\infty}(-1)^k\eta^{(k)}(t) = \nu(\{0\})$.

In order to make the proof of Theorem 2 more transparent we want to separate the influence of the point mass at zero and the integral against the measure. Therefore we write

$$(-1)^k\eta^{(k)}(t) = c + \int_{0^+}^\infty e^{-t\sigma}\, d\nu(\sigma), \quad t > 0,$$

where the integral now definitely does not involve any point mass at zero. This corresponds to splitting η into a polynomial part $q_k(t) = (-1)^k ct^k/k! +$ lower degree terms, and a part $F = \eta - q_k$ which is in $C[0,\infty)$ with $(-1)^k F^{(k)}(t)$ completely monotonic but nonconstant on $(0,\infty)$. By construction we have $\lim_{t\to\infty} F^{(k)}(t) = 0$, and the measure occurring in the Bernstein-Widder representation of $(-1)^k F^{(k)}$ has no point mass at zero. That measure is $\nu - \nu(\{0\})\delta_0$.

Consider now a nonzero compactly supported regular Borel measure μ which annihilates π^n_{k-1}. Then applying Lemma 2 to the polynomial q_k which occurs in the splitting of η,

$$\iint q_k(\|\boldsymbol{x}-\boldsymbol{y}\|^2)\, d\mu(\boldsymbol{x})\, d\mu(\boldsymbol{y}) = (-1)^k \frac{c}{k!} \iint \|\boldsymbol{x}-\boldsymbol{y}\|^{2k}\, d\mu(\boldsymbol{x})\, d\mu(\boldsymbol{y}) \geq 0.$$

That is, $q_k(\|\cdot\|^2)$ and $(-1)^k c\|\cdot\|^{2k}/k!$ are integrally conditionally positive definite of order k, but not strictly so.

For the other part of the splitting, we use calculations identical to those in [71, page 17], except applying Fubini's theorem instead of operating with finite sums, and using Lemma 2 rather than its pointwise analogue.

From above we have

$$(-1)^k F^{(k)}(t) = \int_{0^+}^{\infty} e^{-t\sigma}\, d\nu(\sigma), \quad t > 0, \tag{3.13}$$

and using Taylor's Theorem with integral remainder we can write

$$F(t+\epsilon) = \sum_{l=0}^{k-1} \frac{F^{(l)}(\epsilon)}{l!} t^l + \int_0^t \frac{F^{(k)}(u+\epsilon)}{(k-1)!} (t-u)^{k-1}\, du. \tag{3.14}$$

Combining (3.13) and (3.14), then using Fubini gives

$$\begin{aligned} F(t+\epsilon) &= \sum_{l=0}^{k-1} \frac{F^{(l)}(\epsilon)}{l!} t^l + \int_0^t \frac{(-1)^k}{(k-1)!} \int_{0^+}^{\infty} e^{-(u+\epsilon)\sigma}\, d\nu(\sigma)(t-u)^{k-1}\, du \\ &= \sum_{l=0}^{k-1} \frac{F^{(l)}(\epsilon)}{l!} t^l + \int_{0^+}^{\infty} e^{-\epsilon\sigma} \frac{(-1)^k}{(k-1)!} \int_0^t e^{-u\sigma}(t-u)^{k-1}\, du\, d\nu(\sigma). \end{aligned} \tag{3.15}$$

Now applying Taylor's theorem with integral remainder again in the form

$$g(t) - \sum_{l=0}^{k-1} \frac{g^{(l)}(0)}{l!} t^l = \frac{1}{(k-1)!} \int_0^t g^{(k)}(u)(t-u)^{k-1}\, du,$$

this time to the inner integral, we see

$$\frac{(-1)^k}{(k-1)!} \int_0^t e^{-u\sigma}(t-u)^{k-1}\, du = \frac{1}{\sigma^k} \left(e^{-t\sigma} - \sum_{l=0}^{k-1} \frac{(-t\sigma)^l}{l!} \right). \tag{3.16}$$

Now we substitute (3.16) into (3.15) and set $t = \|\boldsymbol{x} - \boldsymbol{y}\|^2$.

$$F(t+\epsilon) = \sum_{l=0}^{k-1} \frac{F^{(l)}(\epsilon)}{l!} t^l + \int_{0^+}^{\infty} \frac{e^{-\epsilon\sigma}}{\sigma^k} \left(e^{-t\sigma} - \sum_{l=0}^{k-1} \frac{(-t\sigma)^l}{l!} \right) d\nu(s). \tag{3.17}$$

$$\begin{aligned} F(\|\boldsymbol{x}-\boldsymbol{y}\|^2 + \epsilon) = &\sum_{l=0}^{k-1} \frac{F^{(l)}(\epsilon)}{l!} \|\boldsymbol{x}-\boldsymbol{y}\|^{2l} + \\ &\int_{0^+}^{\infty} \frac{e^{-\epsilon\sigma}}{\sigma^k} \left(e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma} - \sum_{l=0}^{k-1} \frac{(-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma)^l}{l!} \right) d\nu(s). \end{aligned} \tag{3.18}$$

Integrating both sides with respect to $\mu(\boldsymbol{x})$ and $\mu(\boldsymbol{y})$, and again changing the order of integration we get

$$\iint F(\|\boldsymbol{x}-\boldsymbol{y}\|^2+\epsilon)\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y}) \\ = \int_{0^+}^{\infty} e^{-\epsilon\sigma}\sigma^{-k}\left\{\iint e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma}\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})\right\}\,d\nu(\sigma), \tag{3.19}$$

where the sum terms vanish due to Lemma 2. Now since $F^{(k)}$ is nonconstant (by construction, page 14), ν is non-constant (by (3.13), page 15), and there exists $a>0$ so that $\int_a^{2a} 1\,d\nu(\sigma)>0$. Also, since $\mu\neq 0$, Lemma 1 implies that the quantity in braces, $\{\ \}$, above is a positive and continuous function of $\sigma>0$. Hence it has a positive lower bound on the compact set $[a,2a]$. Therefore for all sufficiently small $\epsilon>0$,

$$\begin{aligned}\iint F(\|\boldsymbol{x}-\boldsymbol{y}\|^2+\epsilon)\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y}) \\ &\geq \int_a^{2a} e^{-\epsilon\sigma}\sigma^{-k}\left\{\iint e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma}\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})\right\}\,d\nu(\sigma)\\ &> (2a)^{-k}\int_a^{2a} e^{-\epsilon\sigma}\left\{\iint e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma}\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})\right\}\,d\nu(\sigma)\\ &> \frac{1}{2}(2a)^{-k}\int_a^{2a}\left\{\iint e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2\sigma}d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})\right\}\,d\nu(\sigma)\\ &=:\gamma>0.\end{aligned}$$

But from (3.19), $\iint F(\|\boldsymbol{x}-\boldsymbol{y}\|^2+\epsilon)\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})$ is a strictly decreasing function of $\epsilon\in[0,\infty)$. Hence

$$\iint F(\|\boldsymbol{x}-\boldsymbol{y}\|^2)\,d\mu(\boldsymbol{x})\,d\mu(\boldsymbol{y})>\gamma>0,$$

which implies $F(|\cdot|^2)$ is $\mathsf{ISPD}_k(\mathbb{R}^n)$. It follows that $\eta(|\cdot|^2)=q_k(|\cdot|^2)+F(|\cdot|^2)$ is also $\mathsf{ISPD}_k(\mathbb{R}^n)$, the desired result. □

For the sake of completeness we now give a proof of Theorem 1.

Proof of Theorem 1. Consider the case when the right hand side of the linear system (3.3) is zero. Mimicking well known arguments from the pointwise positive definite case, multiply the first row of the block system (3.3) on the left by $\boldsymbol{c}^T$. This yields

$$\mathbf{0}=\boldsymbol{c}^TG\boldsymbol{c}+\boldsymbol{c}^TP\boldsymbol{a}=\boldsymbol{c}^TG\boldsymbol{c}\quad\text{since } P^T\boldsymbol{c}=\mathbf{0}.$$

From the strict conditional positive definiteness of Φ this implies $\boldsymbol{c} = \mathbf{0}$. Substituting back, the first row of the block system becomes $P\boldsymbol{a} = \mathbf{0}$. But $P\boldsymbol{a}$ is a vector whose i-th component is μ_i applied to the polynomial $q = \sum_{j=1}^{\ell} a_j p_j$. Hence the unisolvency implies $\boldsymbol{a} = \mathbf{0}$. Therefore the only solution to the homogeneous equation is the trivial one and the matrix on the left of equation (3.3) is invertible. Hence, there is a unique solution for any given right hand side. □

3.3 Computational Issues

In this section we address some computational issues.

In the Lagrange interpolation setting it is very useful that the unisolvency condition of the appropriate variant of Theorem 1 can be checked very quickly when only linear polynomials are involved. Specifically, a set of point evaluations is unisolvent for π_1^n if and only if there is no single hyperplane containing all the points.

For integral functionals we have the following related sufficient condition:

Lemma 3. *Let $\mathcal{C} = \{\nu_1, \ldots, \nu_m\}$ be a set of $m > n$ linearly independent compactly supported regular Borel measures on $\mathbb{R}^n$. Suppose that there is a subset $\mathcal{B} = \{\mu_1, \ldots, \mu_{n+1}\}$ of $\mathcal{C}$ such that each element in $\mathcal{B}$ is a positive measure. Associate with each μ_i a corresponding connected compact set A_i so that* $\mathrm{supp}(\mu_i) \subset A_i$. *If the sets $\{A_i | 1 \leq i \leq n+1\}$ can be chosen to be disjoint, and such that no one hyperplane intersects them all, then the set of functionals $\mathcal{C}$ is unisolvent for linears on $\mathbb{R}^n$.*

Proof. It suffices to prove that a set $\mathcal{B}$ of $n+1$ measures with the properties listed in the statement of the lemma is unisolvent for linears. We carry out the details in the special case of $\mathbb{R}^2$, the generalisation to $\mathbb{R}^n$ being easy.

Let $\{p_1, p_2, p_3\}$ be a basis for the linears. Then the pointwise interpolation determinant

$$D(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = \begin{vmatrix} p_1(\boldsymbol{x}_1) & p_2(\boldsymbol{x}_1) & p_3(\boldsymbol{x}_1) \\ p_1(\boldsymbol{x}_2) & p_2(\boldsymbol{x}_2) & p_3(\boldsymbol{x}_2) \\ p_1(\boldsymbol{x}_3) & p_2(\boldsymbol{x}_3) & p_3(\boldsymbol{x}_3) \end{vmatrix}$$

is nonzero for any $\boldsymbol{x}_i \in A_i$, since these points are not collinear. Therefore, considering each $\boldsymbol{x}_i$ as a continuous function of t, $\boldsymbol{x}_i = \boldsymbol{x}_i(t)$, $1 \leq i \leq 3$, and noting that connectedness and pathwise connectedness are equivalent in $\mathbb{R}^n$, by the Intermediate Value Theorem this determinant must have constant sign for $\boldsymbol{x}_i \in A_i$. Moreover, since D is continuous, and the A_i are compact, $|D(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)|$ achieves its infimum on $A_1 \times A_2 \times A_3$. Therefore there exists α such that

$$|D(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)| \geq \alpha > 0,$$

whenever $x_i \in A_i$, $1 \leq i \leq 3$. Integrating we find

$$\begin{aligned}\Delta &= \begin{vmatrix} \int p_1(\boldsymbol{x}_1)\,d\mu_1 & \int p_2(\boldsymbol{x}_1)\,d\mu_1 & \int p_3(\boldsymbol{x}_1)\,d\mu_1 \\ \int p_1(\boldsymbol{x}_2)\,d\mu_2 & \int p_2(\boldsymbol{x}_2)\,d\mu_2 & \int p_3(\boldsymbol{x}_2)\,d\mu_2 \\ \int p_1(\boldsymbol{x}_3)\,d\mu_3 & \int p_2(\boldsymbol{x}_3)\,d\mu_3 & \int p_3(\boldsymbol{x}_3)\,d\mu_3 \end{vmatrix} \\ &= \iiint D(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)\,d\mu_1(\boldsymbol{x}_1)d\mu_2(\boldsymbol{x}_2)d\mu_3(\boldsymbol{x}_3) \\ &= \int_{A_1}\int_{A_2}\int_{A_3} D(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)\,d\mu_1(\boldsymbol{x}_1)d\mu_2(\boldsymbol{x}_2)d\mu_3(\boldsymbol{x}_3).\end{aligned}$$

Therefore since D has constant sign,

$$\begin{aligned}|\Delta| &\geq \int_{A_1}\int_{A_2}\int_{A_3} \alpha\,d\mu_1(\boldsymbol{x}_1)d\mu_2(\boldsymbol{x}_2)d\mu_3(\boldsymbol{x}_3) \\ &= \alpha\int_{A_1} d\mu_1(\boldsymbol{x}_1)\int_{A_2} d\mu_2(\boldsymbol{x}_2)\int_{A_3} d\mu_3(\boldsymbol{x}_3) \\ &> 0,\end{aligned}$$

as the μ_i are positive measures with support in the associated A_i. Therefore $\Delta \neq 0$, which implies the required result. □

Again following the point evaluation case, it is useful to replace the linear system (3.3) by a symmetric positive definite one. This allows solution by Cholesky decomposition or by suitable iterative methods, improving speed and stability. We generalise the construction given for the pointwise case in [10].

Our construction below assumes that the functionals $\mu_1,\ldots,\mu_m$ have been reordered if necessary so that the first ℓ are unisolvent for π^n_{k-1}. Begin by choosing Q to be any $m \times (m-\ell)$ matrix whose columns span the orthogonal complement of the column space of P, so $P^TQ = 0$. From the first row of equation (3.3), setting $\boldsymbol{c} = Q\boldsymbol{\gamma}$,

$$GQ\boldsymbol{\gamma} + P\boldsymbol{a} = \boldsymbol{b},$$

implying

$$Q^TGQ\boldsymbol{\gamma} = Q^T\boldsymbol{b},$$

using the property of Q above, or

$$Q^T\left(\boldsymbol{b} - GQ\boldsymbol{\gamma}\right) = \boldsymbol{0},$$

so $\boldsymbol{b} - GQ\boldsymbol{\gamma}$ is in the column space of P. Therefore the system (3.3) can be

solved as in Algorithm 1.

Algorithm 1 Procedure for Solving the Integral Interpolation Problem

Input:

List of integral sources μ_j where $1 \leq j \leq m$,
vector of associated scalar data values $\boldsymbol{b}$ where $b_j = \int f(\boldsymbol{x})\, d\mu_j(\boldsymbol{x})$

Output:

Integral approximation s to f

1: Calculate G and Q.
2: Solve the $(m-\ell)\times(m-\ell)$ positive definite system $\left(Q^T G Q\right)\boldsymbol{\gamma} = Q^T \boldsymbol{b}$ for $\boldsymbol{\gamma}$.
3: Set $\boldsymbol{c} = Q\boldsymbol{\gamma}$. Set $\tilde{s} = \sum_j c_j \int \Phi(\boldsymbol{x} - \boldsymbol{y})\, d\mu_j(\boldsymbol{y})$.
4: Find the $p \in \pi^n_{k-1}$ which integrally interpolates the residual $(f - \tilde{s})$ with respect to the functionals $\mu_1, \ldots, \mu_\ell$. Then $s = p + \tilde{s}$.

It remains to construct a suitable matrix Q. Proceed as follows. Construct $\{p_1, \ldots, p_\ell\} \subset \pi^n_{k-1}$ biorthogonal to $\mu_1, \ldots, \mu_\ell$, that is, satisfying $\mu_i(p_j) = \delta_{ij}$. $\mathcal{L}(g) = \sum_{t=1}^{\ell} \left(\int g(x)\, d\mu_t(x)\right) p_t$ is then a projection onto π^n_{k-1}, and in particular $\mathcal{L}(p_i) = p_i$. $\mathcal{L}$ is the Lagrange polynomial projection for the functionals $\mu_1, \ldots, \mu_\ell$. Let m be the number of centres, and define

$$Q = \begin{bmatrix} \hat{Q} \\ I_{m-l} \end{bmatrix}, \text{ where } \hat{Q}_{l\times(m-l)} = \left[\hat{Q}_{ij}\right] = \left[-\int p_i\, d\mu_{l+j}\right]. \tag{3.20}$$

Q clearly has full rank. $P^T_{ij} = \int p_i(\boldsymbol{x})\, d\mu_j(\boldsymbol{x})$, so the ith row of P^T is

$$\left[\int p_i\, d\mu_1, \int p_i\, d\mu_2, \ldots, \int p_i\, d\mu_m\right].$$

Similarly, the jth column of Q is

$$\left[-\int p_1\, d\mu_{\ell+j}, -\int p_2\, d\mu_{\ell+j}, \ldots, -\int p_\ell\, d\mu_{\ell+j}, 0, \ldots, 0, 1, 0, \ldots, 0\right]^T$$

where the 1 is in the $(\ell + j)$-th position. Hence the ijth element of $P^T Q$ is

$$\begin{aligned}
\int p_i\, d\mu_{\ell+j} - \sum_{t=1}^{\ell} \left(\int p_i\, d\mu_t\right) \int p_t\, d\mu_{\ell+j} &= \int \left\{ p_i - \sum_{t=1}^{\ell} \left(\int p_i\, d\mu_t\right) p_t \right\} d\mu_{\ell+j} \\
&= \int \left\{ p_i - \mathcal{L}(p_i) \right\} d\mu_{\ell+j} \\
&= 0.
\end{aligned}$$

Thus $P^T Q = 0$ as required.

Chapter 4

Explicit Formulations

In this chapter we derive explicit line and ball source basic functions from commonly used point source radial basic functions. These can then be directly applied with the approach described in Section 3.3, and in particular Algorithm 1. While in principle we are simply integrating the parent basic functions over appropriate subdomains, achieving this in practice can require careful parameterisation choices and involve considerable trickery.

4.1 Line Sources

In this section we consider interpolation problems in which the data to be fitted is a mixture of point values and averages over line segments. In view of the formulation given in the introduction we will choose a *parent basic function* Φ and interpolate using a combination of a low degree polynomial and line segment sources derived from Φ.

The (uniform weight) line segment source derived from Φ and corresponding to a line segment $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \subset \mathbb{R}^n$ has the following value at $\boldsymbol{x}$

$$\Psi(\langle \boldsymbol{a}, \boldsymbol{b} \rangle, \boldsymbol{x}) := \frac{1}{\|\boldsymbol{b} - \boldsymbol{a}\|} \int_{\boldsymbol{\xi} \in \langle \boldsymbol{a}, \boldsymbol{b} \rangle} \Phi(\boldsymbol{x} - \boldsymbol{\xi}) \, d\boldsymbol{\xi}.$$

Note that the integral is weighted by the inverse of the length of the interval being integrated over. This normalisation ensures that as the segment shrinks to a point the line segment source converges to the corresponding parent basic function. The normalisation also helps the conditioning of the linear systems (3.3) being used to calculate integral interpolants.

In order to give explicit formulas for some of these line sources we standardise on a geometry as in Figure 4.1. In the diagram d is the perpendicular distance from the evaluation point $\boldsymbol{x}$ to the line through points $\boldsymbol{a}$ and $\boldsymbol{b}$, and $\boldsymbol{p}$ is the

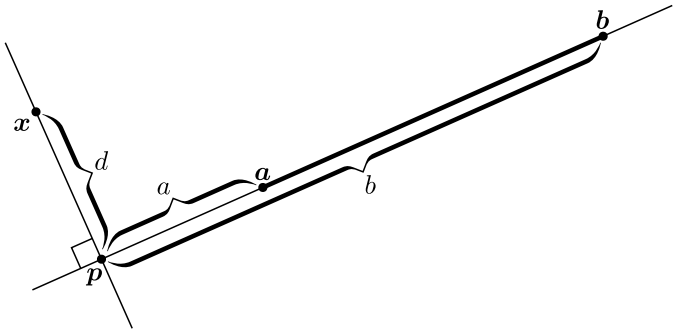


Figure 4.1: Line integral parameters.

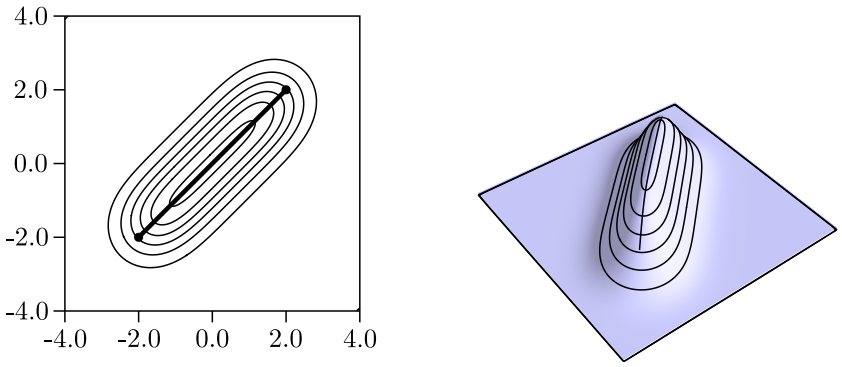
foot-point, the projection of $\boldsymbol{x}$ onto this line. a and b are the signed distances to $\boldsymbol{a}$, respectively $\boldsymbol{b}$, from the foot-point with the direction from $\boldsymbol{a}$ to $\boldsymbol{b}$ taken as positive. The "coordinates" a, b and d are trivial to calculate. Explicitly, the footpoint is given by

$$\boldsymbol{p} = \boldsymbol{a} + \left\{ (\boldsymbol{x} - \boldsymbol{a})^T \boldsymbol{u} \right\} \boldsymbol{u}, \quad \text{where} \quad \boldsymbol{u} = \frac{\boldsymbol{b} - \boldsymbol{a}}{\|\boldsymbol{b} - \boldsymbol{a}\|},$$

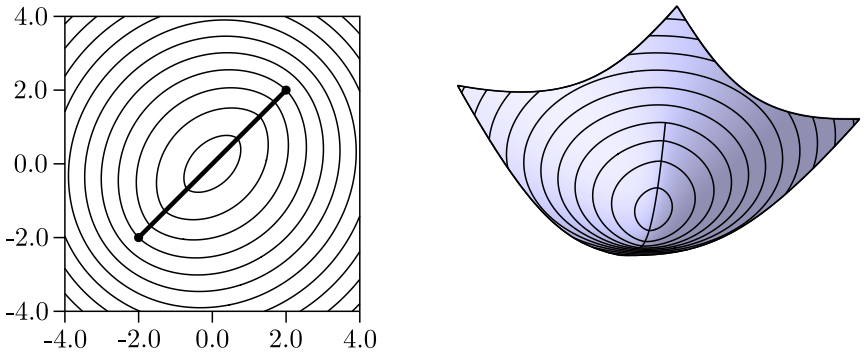
and then

$$a = (\boldsymbol{a} - \boldsymbol{p})^T \boldsymbol{u}, \quad b = (\boldsymbol{b} - \boldsymbol{p})^T \boldsymbol{u}, \quad \text{and} \quad d^2 = (\boldsymbol{x} - \boldsymbol{p})^T (\boldsymbol{x} - \boldsymbol{p}).$$

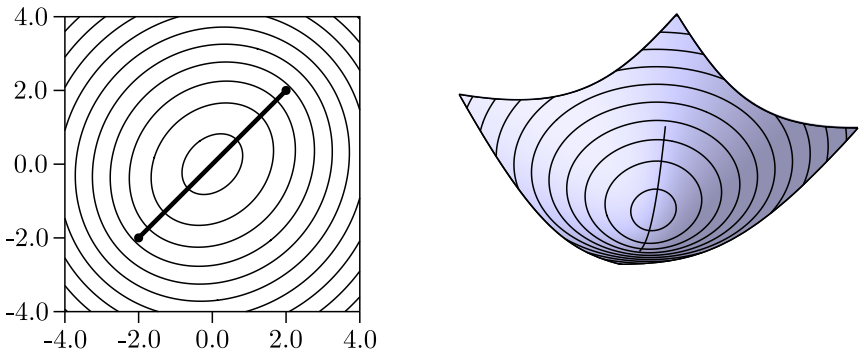
We proceed to give explicit closed forms for various line segment sources. This enables us to evaluate the final fitted function s of (3.1) without any quadrature, and to form the matrix G of the fitting equations (3.3) with only univariate quadrature. Contour and surface plots of these line source basic functions are given in Figures 4.2. In all cases except for the Wendland function, the stated positive definiteness properties follow from Theorem 2, while the positive definiteness properties of the Wendland function follow from Wendland [129] and Iske [51]. The formulas were computed symbolically using Sage [108] (with Maxima [1] under the surface) and SymPy [111], and checked against numerically integrated versions computed using SciPy [55] and its interface to QUADPACK [92].



(a) Gaussian ($\nu = 1$)

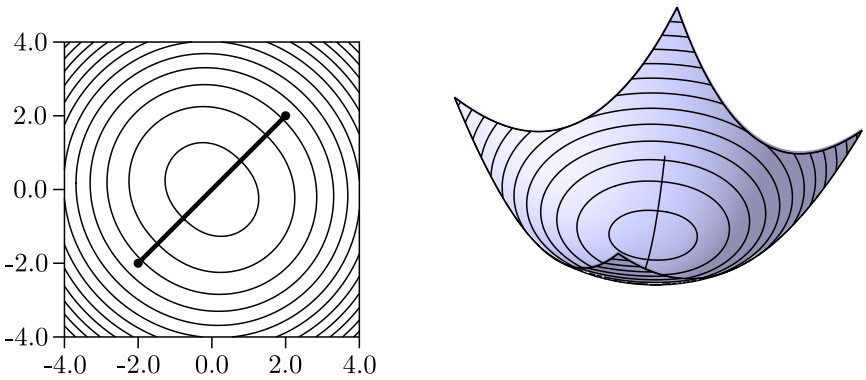


(b) Linear

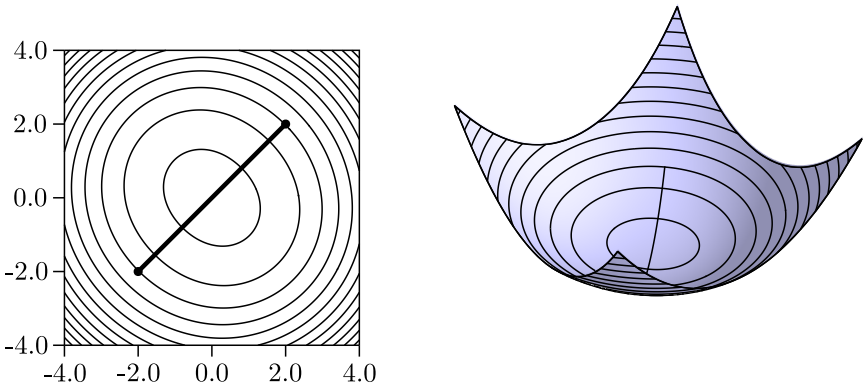


(c) Multiquadric ($c = 1$)

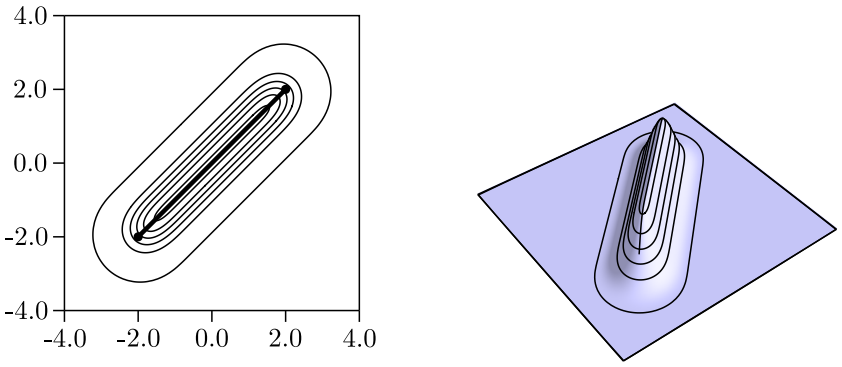
Figure 4.2: Line integral basis functions. Note that the contour spacing varies between the subfigures, as the functions' growth rates differ drastically.



(d) Thin-plate spline



(e) Cubic



(f) Wendland $\Phi_{3,1}$ (support radius 1.5)

Figure 4.2: Line integral basis functions continued.

Gaussian line source

The Gaussian basic function is $\mathsf{ISPD}(\mathbb{R}^n)$ (integrally strictly positive definite on $\mathbb{R}^n$) for all n.

$$\Phi(\boldsymbol{x}) = e^{-\nu^2\|\boldsymbol{x}\|^2}, \quad \boldsymbol{x} \in \mathbb{R}^n,\ \nu > 0.$$

$$\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle \boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) = \frac{\sqrt{\pi}}{2\nu} e^{-\nu^2 d^2}\left(\operatorname{erf}(\nu b) - \operatorname{erf}(\nu a)\right).$$

Linear line source

The negative of the linear basic function is $\mathsf{ISPD}_1(\mathbb{R}^n)$ (integrally strictly conditionally positive definite of order 1 on $\mathbb{R}^n$) for all n. RBFs of the form (3.2) based on this Φ and linear polynomials are biharmonic splines in $\mathbb{R}^3$.

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|, \quad \boldsymbol{x} \in \mathbb{R}^n.$$

$$\begin{aligned}\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle \boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) = {} & \frac{1}{2}\left\{b\sqrt{d^2+b^2} + d^2\ln\left(b+\sqrt{d^2+b^2}\right)\right\} \\ & -\frac{1}{2}\left\{a\sqrt{d^2+a^2} + d^2\ln\left(a+\sqrt{d^2+a^2}\right)\right\}.\end{aligned}$$

Multiquadric line source

The negative of the multiquadric basic function is $\mathsf{ISPD}_1(\mathbb{R}^n)$ for all n.

$$\Phi(\boldsymbol{x}) = \sqrt{\|\boldsymbol{x}\|^2 + c^2}, \quad \boldsymbol{x} \in \mathbb{R}^n,\ c > 0.$$

$$\begin{aligned}&\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle \boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) \\ &\quad = \frac{1}{2}\left\{b\sqrt{d^2+b^2+c^2} + \left(d^2+c^2\right)\ln\left(b+\sqrt{d^2+b^2+c^2}\right)\right\} \\ &\qquad -\frac{1}{2}\left\{a\sqrt{d^2+a^2+c^2} + \left(d^2+c^2\right)\ln\left(a+\sqrt{d^2+a^2+c^2}\right)\right\}.\end{aligned}$$

Thin-plate spline line source

The thin-plate basic function is $\mathsf{ISPD}_2(\mathbb{R}^n)$ for all n. RBFs of the form (3.2) based on this Φ and linear polynomials are biharmonic splines in $\mathbb{R}^2$.

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^2 \ln\|\boldsymbol{x}\|, \quad \boldsymbol{x} \in \mathbb{R}^n.$$

$$\begin{aligned}
&\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle\boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) \\
&\quad= \left\{d^2 b\left(\ln(d^2+b^2)-\frac{4}{3}\right)+\frac{4d^3}{3}\arctan\left(\frac{b}{d}\right)+\frac{b^3}{9}\Big(3\ln\left(d^2+b^2\right)-2\Big)\right\} \\
&\quad\quad-\left\{d^2 a\left(\ln(d^2+a^2)-\frac{4}{3}\right)+\frac{4d^3}{3}\arctan\left(\frac{a}{d}\right)+\frac{a^3}{9}\Big(3\ln\left(d^2+a^2\right)-2\Big)\right\}.
\end{aligned}$$

Cubic line source

The cubic basic function is $\mathsf{ISPD}_2(\mathbb{R}^n)$ for all n. RBFs of the form (3.2) based on this Φ and quadratic polynomials are triharmonic splines in $\mathbb{R}^3$.

$$\Phi(\boldsymbol{x})=\|\boldsymbol{x}\|^3,\quad \boldsymbol{x}\in\mathbb{R}^n.$$

$$\begin{aligned}
&\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle\boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) \\
&\quad= \frac{1}{8}\left\{2b\left(d^2+b^2\right)^{3/2}+3d^2 b\sqrt{d^2+b^2}+3d^4\ln\left(b+\sqrt{d^2+b^2}\right)\right\} \\
&\quad\quad-\frac{1}{8}\left\{2a\left(d^2+a^2\right)^{3/2}+3d^2 a\sqrt{d^2+a^2}+3d^4\ln\left(a+\sqrt{d^2+a^2}\right)\right\}.
\end{aligned}$$

Wendland $\Phi_{3,1}$ line source

The Wendland basic function is $\mathsf{ISPD}(\mathbb{R}^n)$ for $n\le 3$.

$$\Phi(\boldsymbol{x})=\left(1-\frac{\|\boldsymbol{x}\|}{s}\right)_+^4\left(4\frac{\|\boldsymbol{x}\|}{s}+1\right),\quad \boldsymbol{x}\in\mathbb{R}^n,\ n\le 3,$$

where $s>0$ is the radius of support. Here, the notation $(\cdot)_+$ is defined as

$$(a)_+=\begin{cases}0, & a\le 0\\ a, & a>0.\end{cases}$$

If $\boldsymbol{x}$ is further than s units from the line segment then $\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle \boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) = 0$, otherwise

$$
\begin{aligned}
&\|\boldsymbol{b}-\boldsymbol{a}\|\,\Psi(\langle \boldsymbol{a},\boldsymbol{b}\rangle,\boldsymbol{x}) \\
&\quad= \quad \Big\{ b + \tfrac{1}{s^2}\left(-10bd^2 - \tfrac{10b^3}{3}\right) \\
&\qquad + \tfrac{1}{s^3}\left(\left(\tfrac{25bd^2}{2} + 5b^3\right)\sqrt{d^2+b^2} + \tfrac{15d^4}{2}\sinh^{-1}\left(\tfrac{b}{d}\right)\right) \\
&\qquad + \tfrac{1}{s^4}\left(-15bd^4 - 10b^3d^2 - 3b^5\right) \\
&\qquad + \tfrac{1}{s^5}\left(\left(\tfrac{11bd^4}{4} + \tfrac{13b^3d^2}{6} + \tfrac{2b^5}{3}\right)\sqrt{d^2+b^2} + \tfrac{5d^6}{4}\sinh^{-1}\left(\tfrac{b}{d}\right)\right) \Big\} \\
&\quad\quad -\Big\{ a + \tfrac{1}{s^2}\left(-10ad^2 - \tfrac{10a^3}{3}\right) \\
&\qquad + \tfrac{1}{s^3}\left(\left(\tfrac{25ad^2}{2} + 5a^3\right)\sqrt{d^2+a^2} + \tfrac{15d^4}{2}\sinh^{-1}\left(\tfrac{a}{d}\right)\right) \\
&\qquad + \tfrac{1}{s^4}\left(-15ad^4 - 10a^3d^2 - 3a^5\right) \\
&\qquad + \tfrac{1}{s^5}\left(\left(\tfrac{11ad^4}{4} + \tfrac{13a^3d^2}{6} + \tfrac{2a^5}{3}\right)\sqrt{d^2+a^2} + \tfrac{5d^6}{4}\sinh^{-1}\left(\tfrac{a}{d}\right)\right) \Big\}.
\end{aligned}
$$

Observe the change in behaviour in Figure 4.2 between the "long" contour shapes of the line integral basis functions in 4.2a–c and 4.2f, and the "wide" shapes of 4.2d–e. This shape depends on the growth rate of the parent basic function, with the change-over from long to wide occurring at $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$, which has circular contours. (Of course, $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$ is unsuitable as an RBF basic function. No matter how many centres we use the function s stays within the space of quadratic polynomials, which obviously limits the approximation power, and the number of interpolation conditions it can satisfy.)

4.2 Ball Sources

In this section we develop explicit formulas for ball sources in $\mathbb{R}^3$ and $\mathbb{R}^5$. Our first motivation is to estimate densities in point clouds by performing integral interpolation to point counts over spheres. For example, in three dimensions the data could be particles in a fluid, or in five dimensions they could be samples in a photon map (three for spatial location and two for incidence angle). For the latter application, it is important to have compactly supported basic functions available, since the photon samples represent *all* the light in the scene, and empty spaces signify darkness rather than missing information (i.e. extrapolation should always go to zero). Such interpolation should likewise be useful

in extracting low frequency trends from noisy data by fitting to average values over spheres.

The ball source double integrals have proven much more tractable than those of the line sources, and we have found analytic solutions for all the parent basic functions presented here, and several others (though some of them are too long to be reproduced in full). When performing integral interpolation with the following ball source basic functions there is no need for any numerical integration.

We proceed to develop the formulas. Let $\Phi(\boldsymbol{x})$ be our radial parent basic function, and define $\mathcal{B}_c(\boldsymbol{x})$, the normalised characteristic function of the sphere with radius c, center the origin, as follows.

$$\mathcal{B}_c(\boldsymbol{x}) = \begin{cases} \dfrac{3}{4\pi c^3}, & \|\boldsymbol{x}\| \leq c, \\ 0, & \|\boldsymbol{x}\| > c. \end{cases} \qquad \text{for } \mathbb{R}^3$$

$$\mathcal{B}_c(\boldsymbol{x}) = \begin{cases} \dfrac{15}{8\pi^2 c^5}, & \|\boldsymbol{x}\| \leq c, \\ 0, & \|\boldsymbol{x}\| > c. \end{cases} \qquad \text{for } \mathbb{R}^5$$

Clearly the integrals of these functions over the appropriate $\mathbb{R}^n$ are 1. Note that the normalised integral of Φ, centred at $\boldsymbol{x}$, over $\operatorname{supp}(\mathcal{B}_c)$, centred at the origin, is equal to the value of $\Phi \star \mathcal{B}_c$ at $\boldsymbol{x}$:

$$\frac{1}{\operatorname{vol}(\mathcal{B}_c)} \int_{\operatorname{supp}(\mathcal{B}_c)} \Phi(\boldsymbol{\xi}-\boldsymbol{x})\, d\boldsymbol{\xi} = \int \Phi(\boldsymbol{\xi}-\boldsymbol{x})\mathcal{B}_c(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \int \Phi(\boldsymbol{x}-\boldsymbol{\xi})\mathcal{B}_c(\boldsymbol{\xi})\, d\boldsymbol{\xi} = (\Phi \star \mathcal{B}_c)(\boldsymbol{x}),$$

where we have used the evenness of Φ. For the remainder of this thesis, we refer to ball source functions as integrals or convolutions somewhat interchangeably, depending on context. Note that the convolution of two radial functions is again radial; see for example [8, Lemma 5.1].

Referring to Problems 1 and 2, we need to construct a basic function from the integral of Φ over $\operatorname{supp}(\mathcal{B}_c)$, and then integrate this over another ball $\operatorname{supp}(\mathcal{B}_d)$, where d is possibly different from c. Equivalently, we need to find $(\Phi \star \mathcal{B}_c)(\boldsymbol{x})$ and $(\Phi \star \mathcal{B}_c \star \mathcal{B}_d)(\boldsymbol{x})$. Since the case where $d = c$ is of practical use and often leads to much simpler formulas, we also calculate $(\Phi \star \mathcal{B}_c \star \mathcal{B}_c)(\boldsymbol{x})$ explicitly.

Ball sources made from the convolutions $(\Phi \star \mathcal{B}_c)(\boldsymbol{x})$ can usually be calculated explicitly when Φ is radial, and we have two methods for deriving these formulas. In [9], operators are used to convert 3-dimensional convolutions to more tractable 1-dimensional convolutions, and we present this approach first here. However, we have also developed a simpler method based on direct integration with spherical shells, presented next. Both methods work by reducing the multidimensional integral to a 1-dimensional one.

4.2.1 The Operator Approach

We use the operators

$$(If)(r) = \int_r^\infty sf(s)\,ds, \qquad (Dg)(r) = -\frac{1}{r}\frac{dg}{dr}, \qquad r \geq 0,$$

which satisfy

$$f \star_{n+2} g = 2\pi D(If \star_n Ig), \tag{4.1}$$

for compactly supported bounded radial functions f and g. Here, we use the notation f, g both for the even functions of one variable $f(r)$, $g(r)$, and also for the radial functions of several variables $f(\|\boldsymbol{x}\|)$ and $g(\|\boldsymbol{x}\|)$, with $\boldsymbol{x} \in \mathbb{R}^n$. Like f and g, $(If)(r)$ and $(Dg)(r)$ are continued by evenness for $r < 0$. The subscript on the convolution symbol $\star$ denotes the dimension in which the convolution is performed. Thus $f \star_{n+2} g$ denotes the convolution in $\mathbb{R}^{n+2}$ of the radial functions of $n+2$ variables $f(\|\boldsymbol{x}\|)$ and $g(\|\boldsymbol{x}\|)$.

In the approximation theory context these formulas were used by Wu [134] and Wendland [127], and developed fully by Schaback and Wu [102]. However, they had been previously discovered in the geostatistical context by Matheron [69]. See Chiles and Delfiner [20] for references to relevant geostatistical literature.

In order to use these formulas on non compactly supported functions we create a compactly supported version by truncating the function at an arbitrary distance γ, chosen to always be far enough out that the truncation does not affect us. We can always determine this distance as the I operator preserves compact supports exactly. Further, it is straightforward to show that for a truncated function $\tilde{f}$ defined like this, γ will only appear in $I\tilde{f} \star_1 I\mathcal{B}_c$ as part of constant terms, and will thus vanish completely when D is applied.

As a concrete example, we show the calculation of $\Phi \star_3 \mathcal{B}_c$ for $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$. From (4.1) we have

$$\Phi \star_3 \mathcal{B}_c = 2\pi D(I\Phi \star_1 I\mathcal{B}_c),$$

so we first need to find $I\Phi$ and $I\mathcal{B}_c$. Define

$$\tilde{\Phi}(\boldsymbol{x}) = \begin{cases} \Phi(\boldsymbol{x}), & \|\boldsymbol{x}\| < \gamma, \\ 0, & \|\boldsymbol{x}\| \geq \gamma. \end{cases}$$

Then for $0 \leq r < \gamma$ and $0 \leq r < c$ respectively,

$$\begin{aligned}(I\tilde{\Phi})(r) &= \int_r^\infty s\tilde{\Phi}(s)\,ds & (I\mathcal{B}_c)(r) &= \int_r^\infty s\mathcal{B}_c(s)\,ds \\ &= \int_r^\gamma s^2\,ds & &= \int_r^c s\frac{3}{4\pi c^3}\,ds \\ &= \frac{\gamma^3}{3} - \frac{r^3}{3}, & &= \frac{3}{4\pi c^3}\left(\frac{c^2}{2} - \frac{r^2}{2}\right).\end{aligned}$$

Next, we find the 1-dimensional convolution,

$$(I\Phi \star_1 I\mathcal{B}_c)(r) = \int_{-\infty}^\infty (I\tilde{\Phi})(\rho)(I\mathcal{B}_c)(r-\rho)\,d\rho. \tag{4.2}$$

As we are at a distance $r \geq 0$ from the origin and the ball has radius c, the limits of the convolution integral will contract to $r - c$ and $r + c$. We choose $\gamma > r + c$ so the truncation is safely out of the way, but we have no such control over r. This will cause trouble when $r < c$ as the formula for $(I\tilde{\Phi})(\rho)$, the first factor of the integrand in (4.2), changes to $(I\tilde{\Phi})(-\rho)$ for $\rho < 0$. We therefore split the problem into two cases, $0 \leq r < c$ and $c \leq r$. (Note that since $(I\mathcal{B}_c)(r)$ is even, this difficulty does not occur with the second factor of the integrand, $(I\mathcal{B}_c)(r-\rho)$.) For the simpler case, $c \leq r$, we have

$$\begin{aligned}(I\Phi \star_1 I\mathcal{B}_c)(r) &= \frac{3}{4\pi c^3}\int_{r-c}^{r+c}\left(\frac{\gamma^3}{3} - \frac{\rho^3}{3}\right)\left(\frac{c^2}{2} - \frac{(r-\rho)^2}{2}\right)d\rho \\ &= \frac{3}{4\pi c^3}\left(\frac{2c^3\gamma^3}{9} - \frac{2c^3r^3}{9} - \frac{2c^5r}{15}\right)\end{aligned}$$

and

$$\begin{aligned}2\pi\left(D(I\Phi \star_1 I\mathcal{B}_c)\right)(r) &= 2\pi\frac{3}{4\pi c^3}\frac{-1}{r}\frac{d}{dr}\left(\frac{2c^3\gamma^3}{9} - \frac{2c^3r^3}{9} - \frac{2c^5r}{15}\right) \\ &= \frac{-3}{2rc^3}\left(\frac{-2c^3r^2}{3} - \frac{2c^5}{15}\right) \\ &= \frac{c^2}{5r} + r.\end{aligned}$$

For $r < c$ we deal with the radial nature of $I\Phi$ by breaking the convolution

integral at 0 as follows:

$$\begin{aligned}(I\Phi \star_1 I\mathcal{B}_c)(r) &= \int_{-\infty}^{\infty} (I\tilde{\Phi})(\rho)(I\mathcal{B}_c)(r-\rho)\,d\rho \\ &= \int_0^{c-r} (I\tilde{\Phi})(\rho)(I\mathcal{B}_c)(r+\rho)\,d\rho + \int_0^{r+c} (I\tilde{\Phi})(\rho)(I\mathcal{B}_c)(r-\rho)\,d\rho\end{aligned}$$

The process then continues in the same manner as before, leaving us with

$$2\pi\left(D(I\Phi \star_1 I\mathcal{B}_c)\right)(r) = \frac{3}{4}c + \frac{r^2}{2c} - \frac{r^4}{20c^3}.$$

Our final result is then

$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \begin{cases} \frac{3c}{4} + \frac{\|\boldsymbol{x}\|^2}{2c} - \frac{\|\boldsymbol{x}\|^4}{20c^3}, & 0 \le \|\boldsymbol{x}\| < c \\ \frac{c^2}{5\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & c \le \|\boldsymbol{x}\|. \end{cases}$$

4.2.2 The Spherical Shell Approach

Alternately, by arranging the integrals to be in terms of spherical shells, we can do them directly. We introduce our notation in Figure 4.3. Given a ball source made from the parent basic function Φ and the ball $\mathcal{B}_c$ (centred at the origin), to evaluate the ball source at $\boldsymbol{x}$, we centre our Φ at $\boldsymbol{x}$ and integrate this over $\text{supp}(\mathcal{B}_c)$. Note that this is identical to centering Φ at a movable point $\boldsymbol{\xi}$ inside $\text{supp}(\mathcal{B}_c)$ and integrating the value of Φ at $\boldsymbol{x}$ as $\boldsymbol{\xi}$ takes every position inside $\text{supp}(\mathcal{B}_c)$—both are $\int \Phi(\boldsymbol{\xi} - \boldsymbol{x})\,dV$, where V is a volume element of appropriate dimension.

Within the ball are isocontours of $\Phi(\boldsymbol{\xi} - \boldsymbol{x})$, forming spherical shells centred at $\boldsymbol{x}$. If we can determine the surface areas of the parts of these shells inside the ball, we can reduce this problem to a 1-dimensional integral in ρ with limits of integration $r \pm c$, where $r = \|\boldsymbol{x}\| \ge 0$. The final integral result will be a function of $r = \|\boldsymbol{x}\|$; it will be radial and centred at the origin (the centre of $\mathcal{B}_c$).

When $r < c$, some of the shells will be complete spheres, so their surface areas are well known and will be different from the incomplete shells, leading to a piecewise result. The 3- and 5-dimensional sphere surface areas are $4\pi\rho^2$ and $\frac{8}{3}\pi^2\rho^4$ respectively.

The incomplete shells are only slightly more complicated. Finding the intersections between the two circles in Figure 4.3 gives us

$$l = \frac{\rho^2 + r^2 - c^2}{2r}.$$

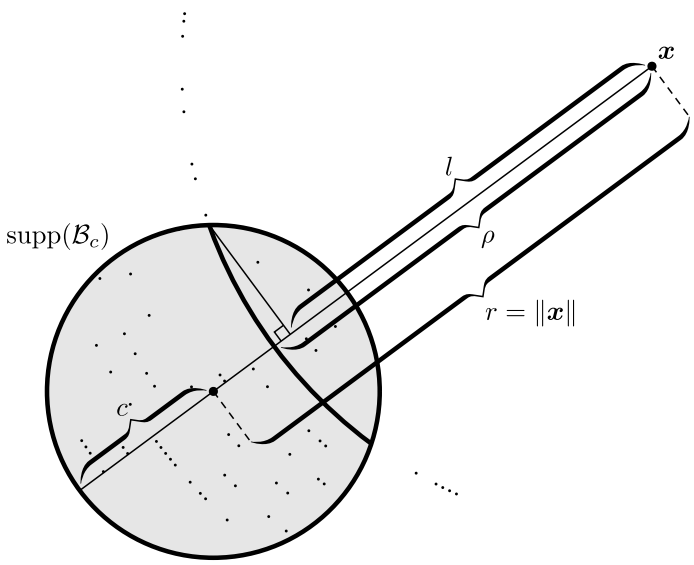


Figure 4.3: Ball integral parameters. The ball $\mathcal{B}_c$ is centred at the origin, and within it are isocontours of $\Phi(\boldsymbol{\xi} - \boldsymbol{x})$.

The surface area is dimension dependent, and can be found with a straightforward surface of revolution integral (but see the end of this section for problems encountered in even dimensions). In the 3-dimensional case, the surface area is $2\pi\left(\rho^2 - \rho l\right)$, and in the 5-dimensional case it is $\frac{2\pi^2}{3}\left(2\rho^4 - 3\rho^3 l + l^3\right)$, leading to the final integrals for $r < c$, combining the sphere and shell parts,

$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \frac{3}{c^3}\int_0^{c-r} \Phi(\rho)\rho^2\, d\rho + \frac{3}{2c^3}\int_{c-r}^{r+c} \Phi(\rho)(\rho^2 - \rho l)\, d\rho, \tag{4.3}$$

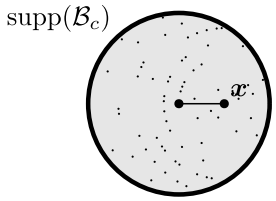
$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \frac{5}{c^5}\int_0^{c-r} \Phi(\rho)\rho^4\, d\rho + \frac{5}{4c^5}\int_{c-r}^{r+c} \Phi(\rho)\left(2\rho^4 - 3\rho^3 l + l^3\right)\, d\rho, \tag{4.4}$$

and for $r \geq c$,

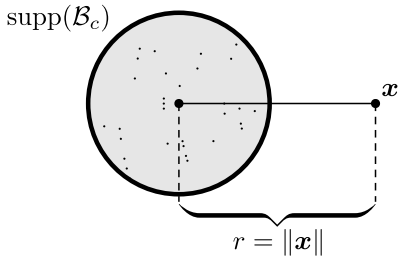
$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \frac{3}{2c^3}\int_{r-c}^{r+c} \Phi(\rho)(\rho^2 - \rho l)\, d\rho, \tag{4.5}$$

$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \frac{5}{4c^5}\int_{r-c}^{r+c} \Phi(\rho)\left(2\rho^4 - 3\rho^3 l + l^3\right)\, d\rho. \tag{4.6}$$

With Φ as our parent basic function, we get $\Phi \star \mathcal{B}_c$, and using this as Φ and feeding it back into the same integral gives us the double convolutions $\Phi \star \mathcal{B}_c \star \mathcal{B}_c$ and $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$ as required for Problem 2.



(a) $0 \leq \|\boldsymbol{x}\| < c$



(b) $c \leq \|\boldsymbol{x}\|$

Figure 4.4: Piecewise sections for $\Phi \star \mathcal{B}_c$, where Φ has global support. The dotted curves are contours of the (shifted) radial function $\Phi(\|\boldsymbol{\xi} - \boldsymbol{x}\|)$ centred at $\boldsymbol{x}$. The dark circle is the boundary of the support of the ball $\mathcal{B}_c$, centred at the origin.

As an example, we recalculate $\Phi \star_3 \mathcal{B}_c$ for $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$. Again, we immediately have two parts, one for $r < c$, where $\boldsymbol{x}$ is inside $\text{supp}(\mathcal{B}_c)$, and the other for $r \geq c$, where it is outside (see Figure 4.4). Looking first at the simpler $r \geq c$ case, from (4.5) we have

$$\begin{aligned}
(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) &= \frac{3}{2c^3} \int_{r-c}^{r+c} \Phi(\rho)(\rho^2 - \rho l)\, d\rho \\
&= \frac{3}{2c^3} \int_{r-c}^{r+c} \rho \left(\rho^2 - \rho \frac{\rho^2 + r^2 - c^2}{2r} \right) d\rho \\
&= \frac{3}{2c^3} \left[\frac{\rho^4}{4} - \frac{1}{2r} \left(\frac{\rho^5}{5} + \frac{\rho^3}{3}(r^2 - c^2) \right) \right]_{r-c}^{r+c} \\
&= \frac{3}{2c^3} \left(\frac{2c^5}{15r} + \frac{2c^3 r}{3} \right) \\
&= \frac{c^2}{5r} + r,
\end{aligned}$$

as before. For $r < c$, from (4.3) we split the integral again, but this time into a sphere part and a shell part, giving

$$\begin{aligned}
(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) &= \frac{3}{c^3} \int_0^{c-r} \Phi(\rho)\rho^2\, d\rho + \frac{3}{2c^3} \int_{c-r}^{r+c} \Phi(\rho)(\rho^2 - \rho l)\, d\rho \\
&= \frac{3}{c^3} \int_0^{c-r} \rho^3\, d\rho + \frac{3}{2c^3} \int_{c-r}^{r+c} \rho(\rho^2 - \rho \frac{\rho^2 + r^2 - c^2}{2r})\, d\rho \\
&= \frac{3}{c^3} \left(\left[\frac{\rho^4}{4} \right]_0^{c-r} + \frac{1}{2} \left[\frac{\rho^4}{4} - \frac{1}{2r} \left(\frac{\rho^5}{5} + \frac{\rho^3}{3}(r^2 - c^2) \right) \right]_{r-c}^{r+c} \right) \\
&= \frac{3}{c^3} \left(\frac{c^4}{4} + \frac{c^2 r^2}{6} - \frac{r^4}{60} \right) \\
&= \frac{3}{4}c + \frac{r^2}{2c} - \frac{r^4}{20c^3},
\end{aligned}$$

as expected.

Since $\Phi \star \mathcal{B}_c$ is piecewise, repeating the procedure to get $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$ becomes slightly more complicated. The situation breaks into two cases, $\frac{c}{2} \leq d < c$ and $d < \frac{c}{2}$, each producing a function with four piecewise sections. This is shown in Figure 4.5, which we explain in detail now.

For the first convolution we had

$$\Phi \star \mathcal{B}_c = \int_{\text{supp}(\mathcal{B}_c)} \Phi(\boldsymbol{\xi} - \boldsymbol{x})\, d\boldsymbol{\xi},$$

and for the second one we have

$$\begin{aligned}\Phi \star \mathcal{B}_c \star \mathcal{B}_d &= (\Phi \star \mathcal{B}_c) \star \mathcal{B}_d \\ &= \int_{\text{supp}(\mathcal{B}_d)} (\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})\, d\boldsymbol{\xi}, \end{aligned} \tag{4.7}$$

which is of the same form as the single convolution, but convolving the function $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi})$ with the ball instead of $\Phi(\boldsymbol{\xi})$, and with a different letter for the ball's radius. As we have already shown, this can be turned into a 1-dimensional integral which can be calculated by being broken into pieces (Equations (4.3) and (4.5)).

The result of the convolution is a radial function of $\|\boldsymbol{x}\|$. It is piecewise, with the boundaries between sections occurring where there are changes in the form of the convolution integral due to interactions between the distance to $\boldsymbol{x}$, the support radius of the ball we are convolving against, and piecewise boundaries in the function we are convolving. We can find these piecewise boundaries by centering our function at a point $\boldsymbol{x}$ (i.e. $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$), starting with $\boldsymbol{x}$ at the origin and moving it away, writing down the transition distances as we meet them.

For the $\frac{c}{2} \leq d < c$ case shown in the left column of Figure 4.5, in (a) we begin with $\boldsymbol{x}$ at the origin, so the left boundary of this section is 0. As we slide $\boldsymbol{x}$ to the right, the convolution integral form remains static until we reach $\|\boldsymbol{x}\| = c - d$ and the left side of the thin circle of radius c meets the left side of the dark circle representing $\text{supp}(\mathcal{B}_d)$.

Up to this point, the only part of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$ inside $\mathcal{B}_d$ (and thus affecting the convolution in (4.7)) has been the part derived from (4.3). For $\|\boldsymbol{\xi} - \boldsymbol{x}\| \geq c$, the definition of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$ changes to one derived from (4.5), and this transition is represented by the thin circle of radius c. At $\|\boldsymbol{x}\| = c - d$, this boundary begins to cross into $\text{supp}(\mathcal{B}_d)$, and so both parts of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$ will contribute to the integral for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$. This means the right boundary of the section shown in (a) is $c - d$, and the section is defined by $0 \leq \|\boldsymbol{x}\| < c - d$.

Now, moving on to (c), $c - d$ becomes the left boundary of this next section. The form of the convolution integral remains the same as we continue to slide $\boldsymbol{x}$ to the right, until we reach $\|\boldsymbol{x}\| = d$. Up to this point the contours of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$ inside $\text{supp}(\mathcal{B}_d)$ have included both complete spheres and partial spherical shells (i.e. the convolution integral is derived from (4.3)), but once $\boldsymbol{x}$ exits $\text{supp}(\mathcal{B}_d)$ there will be no more complete spheres and the form of the integral will change to one derived from (4.5).

In (e), we have a convolution integral with partial spherical shells only, but which still involves contributions from both piecewise sections of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$.

This situation will continue until we reach $\|\boldsymbol{x}\| = c + d$, and the break point of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$ at radius c no longer passes through $\text{supp}(\mathcal{B}_d)$. In (g), the convolution integral involves only the outer section of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$.

For the case in the right column of Figure 4.5, $d < \frac{c}{2}$, the situation is much the same, but the transition points $c - d$ and d are encountered in the opposite order, as $\text{supp}(\mathcal{B}_d)$ is small enough to fit between $\boldsymbol{x}$ and the break point at distance c from $\boldsymbol{x}$.

For the global parent basic functions we have tried (all non-piecewise), the formulas for the two cases come out the same; two adjacent pieces turn out to be equal and can be merged, leaving us with a piecewise function with one fewer piece than Figure 4.5 would indicate. We do not yet know if this will always be the case.

When we move on to deriving ball source formulas from compactly supported basic functions, things become more complicated, since the basic functions are defined piecewise. The support radius is usually defined to be 1, simplifying some expressions but requiring the implicit scaling of all lengths. We choose instead to explicitly use support radius s, allowing it to be directly compared to and combined with an unscaled ball radius c.

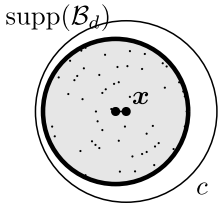
For the first convolution, $\Phi \star \mathcal{B}_c$, there are two cases, depending on which of c or s is greater. These are both shown in Figure 4.6, but we give formulas only for the $c > s$ case.

The sizes of the resulting formulas begin to get out of hand now, and when we move on to $\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_d$ things become very unwieldy. The formulas grow tremendously and the interactions between the different support radii force us to divide into many separate cases. Without loss of generality, we let $c > d$. Then, temporarily fixing $s = 1$, we show the regions of the 30 cases in the c–d plane in Figure 4.7.

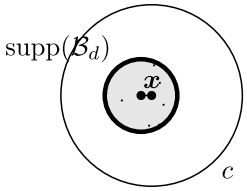
Algebraic representations of these regions are given in Table 4.1. These are clearly not the simplest descriptions of the regions, but the columns of Table 4.1 are the break points of the final piecewise result of the double convolution, in the order they are met (with an implicit $c + s + d$ at the end of each). To help clarify the situation, we show the pieces produced for two particular cases, F in Figure 4.8 and Q in Figure 4.9.

Unfortunately, we have been unable to apply either the operator approach or the shell approach in even dimensions, as the resulting integrals have a different form which has thus far proven intractable. The two dimensional case is particularly appealing, but the four dimensional case will also have practical applications. This remains an important area for future research.

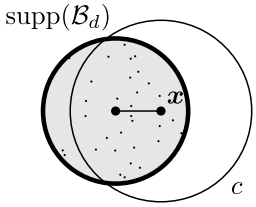
Both of the above methods lead to the same analytic formulations, which we present in Appendix A. They are radial functions centred on the origin which,



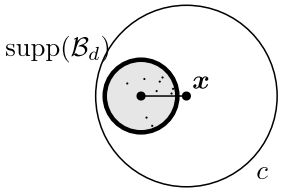
(a) $0 \leq \|\boldsymbol{x}\| < c - d$



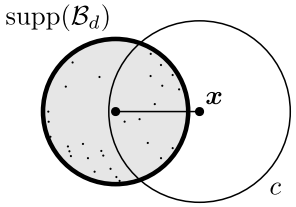
(b) $0 \leq \|\boldsymbol{x}\| < d$



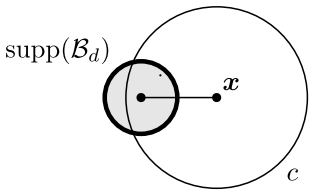
(c) $c - d \leq \|\boldsymbol{x}\| < d$



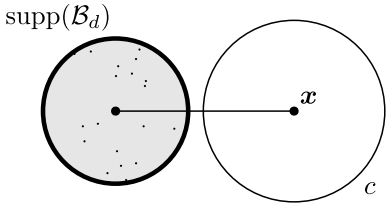
(d) $d \leq \|\boldsymbol{x}\| < c - d$



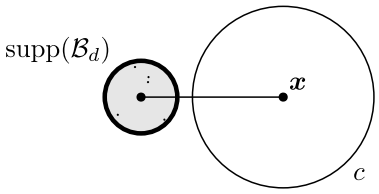
(e) $d \leq \|\boldsymbol{x}\| < c + d$



(f) $c - d \leq \|\boldsymbol{x}\| < c + d$

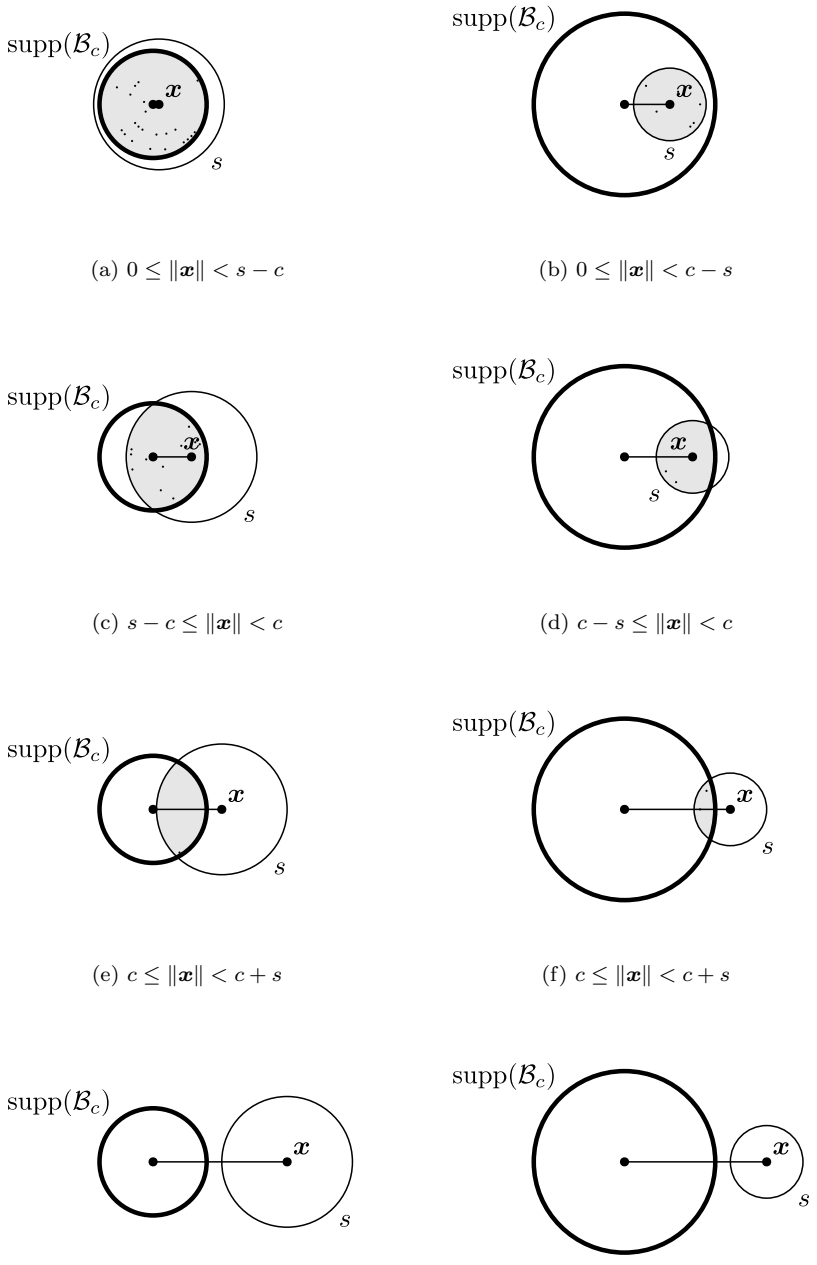


(g) $c + d \leq \|\boldsymbol{x}\|$



(h) $c + d \leq \|\boldsymbol{x}\|$

Figure 4.5: Piecewise sections for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$, where Φ has global support. The left column is for $\frac{c}{2} \leq d < c$, and the right column is for $d < \frac{c}{2}$. The dotted curves are contours of $(\Phi \star \mathcal{B}_c)(\boldsymbol{\xi} - \boldsymbol{x})$, whose definition changes at $\|\boldsymbol{\xi} - \boldsymbol{x}\| = c$ (the thin circle). The dark circle is the boundary of the support of $\mathcal{B}_d$, centred at the origin.



(a) $0 \leq \|\boldsymbol{x}\| < s - c$

(b) $0 \leq \|\boldsymbol{x}\| < c - s$

(c) $s - c \leq \|\boldsymbol{x}\| < c$

(d) $c - s \leq \|\boldsymbol{x}\| < c$

(e) $c \leq \|\boldsymbol{x}\| < c + s$

(f) $c \leq \|\boldsymbol{x}\| < c + s$

(g) $c + s \leq \|\boldsymbol{x}\|$

(h) $c + s \leq \|\boldsymbol{x}\|$

Figure 4.6: Piecewise sections for $\Phi \star \mathcal{B}_c$, where Φ has compact support. The left column is for $c \leq s$, and the right column is for $c > s$.

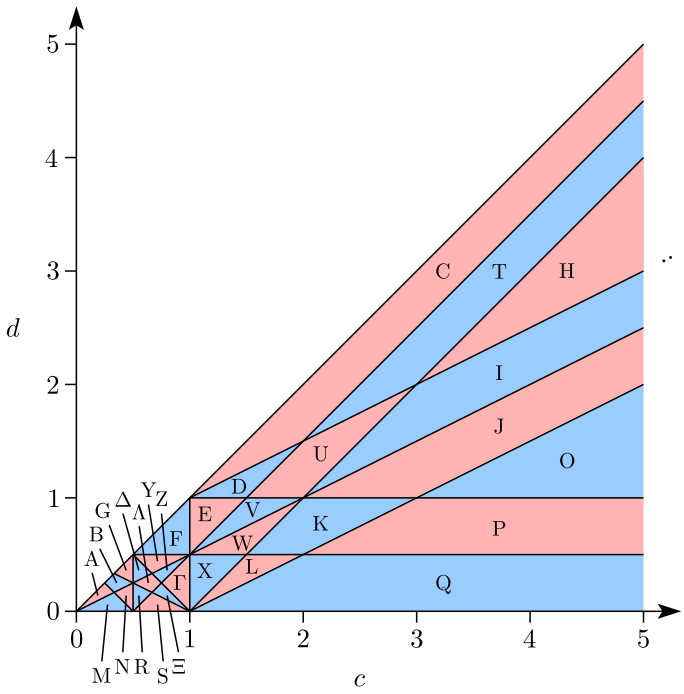
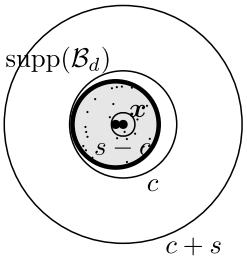


Figure 4.7: Cases for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$, where Φ has compact support $s = 1$.

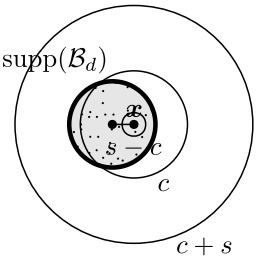
as we have seen, are usually piecewise. The formulas were computed symbolically using Sage [108] (and Maxima [1] under the surface), and checked against numerically integrated versions computed using SciPy [55] and its interface to QUADPACK [92].

A: $0 \leq$	$c-d$	$\leq$	d	$\leq$	$c+d$	$\leq s-c-d \leq$	$s-c+d$	$\leq$	$c+s-d$
B: $0 \leq$	$c-d$	$\leq$	d	$\leq s-c-d \leq$		$c+d$	$\leq s-c+d \leq$		$c+s-d$
C: $0 \leq$	$c-d$	$\leq d-(c-s) \leq$		$c+s-d \leq$		d	$\leq c-s+d \leq$		$c+d$
D: $0 \leq$	$c-d$	$\leq d-(c-s) \leq$		d	$\leq$	$c+s-d \leq$	$c-s+d$	$\leq$	$c+d$
E: $0 \leq$	$c-d$	$\leq d-(c-s) \leq$		d	$\leq$	$c-s+d \leq$	$c+s-d$	$\leq$	$c+d$
F: $0 \leq$	$c-d$	$\leq d-(s-c) \leq$		d	$\leq$	$s-c+d \leq$	$c+s-d$	$\leq$	$c+d$
G: $0 \leq$	$c-d$	$\leq s-c-d \leq$		d	$\leq$	$c+d$	$\leq s-c+d \leq$		$c+s-d$
H: $0 \leq c-s-d \leq$		$c-d$	$\leq$	$c+s-d \leq$		d	$\leq$	$c-s+d \leq$	$c+d$
I: $0 \leq c-s-d \leq$		$c-d$	$\leq$	d	$\leq$	$c+s-d \leq$	$c-s+d$	$\leq$	$c+d$
J: $0 \leq c-s-d \leq$		d	$\leq$	$c-d$	$\leq$	$c+s-d \leq$	$c-s+d$	$\leq$	$c+d$
K: $0 \leq c-s-d \leq$		d	$\leq$	$c-d$	$\leq$	$c-s+d \leq$	$c+s-d$	$\leq$	$c+d$
L: $0 \leq c-s-d \leq$		d	$\leq$	$c-s+d \leq$		$c-d$	$\leq$	$c+d$	$\leq c+s-d$
M: $0 \leq$	d	$\leq$	$c-d$	$\leq$	$c+d$	$\leq s-c-d \leq$	$s-c+d$	$\leq$	$c+s-d$
N: $0 \leq$	d	$\leq$	$c-d$	$\leq s-c-d \leq$		$c+d$	$\leq s-c+d \leq$		$c+s-d$
O: $0 \leq$	d	$\leq c-s-d \leq$		$c-d$	$\leq$	$c+s-d \leq$	$c-s+d$	$\leq$	$c+d$
P: $0 \leq$	d	$\leq c-s-d \leq$		$c-d$	$\leq$	$c-s+d \leq$	$c+s-d$	$\leq$	$c+d$
Q: $0 \leq$	d	$\leq c-s-d \leq$		$c-s+d \leq$		$c-d$	$\leq$	$c+d$	$\leq c+s-d$
R: $0 \leq$	d	$\leq s-c-d \leq$		$c-d$	$\leq$	$s-c+d \leq$	$c+d$	$\leq$	$c+s-d$
S: $0 \leq$	d	$\leq s-c-d \leq$		$s-c+d \leq$		$c-d$	$\leq$	$c+d$	$\leq c+s-d$
T: $0 \leq d-(c-s) \leq$		$c-d$	$\leq$	$c+s-d \leq$		d	$\leq$	$c-s+d \leq$	$c+d$
U: $0 \leq d-(c-s) \leq$		$c-d$	$\leq$	d	$\leq$	$c+s-d \leq$	$c-s+d$	$\leq$	$c+d$
V: $0 \leq d-(c-s) \leq$		$c-d$	$\leq$	d	$\leq$	$c-s+d \leq$	$c+s-d$	$\leq$	$c+d$
W: $0 \leq d-(c-s) \leq$		d	$\leq$	$c-d$	$\leq$	$c-s+d \leq$	$c+s-d$	$\leq$	$c+d$
X: $0 \leq d-(c-s) \leq$		d	$\leq c-s+d \leq$		$c-d$	$\leq$	$c+d$	$\leq$	$c+s-d$
Y: $0 \leq d-(s-c) \leq$		$c-d$	$\leq$	d	$\leq$	$s-c+d \leq$	$c+d$	$\leq$	$c+s-d$
Z: $0 \leq d-(s-c) \leq$		d	$\leq$	$c-d$	$\leq$	$s-c+d \leq$	$c+d$	$\leq$	$c+s-d$
Γ: $0 \leq d-(s-c) \leq$		d	$\leq s-c+d \leq$		$c-d$	$\leq$	$c+d$	$\leq$	$c+s-d$
Δ: $0 \leq s-c-d \leq$		$c-d$	$\leq$	d	$\leq$	$s-c+d \leq$	$c+d$	$\leq$	$c+s-d$
Λ: $0 \leq s-c-d \leq$		d	$\leq$	$c-d$	$\leq$	$s-c+d \leq$	$c+d$	$\leq$	$c+s-d$
Ξ: $0 \leq s-c-d \leq$		d	$\leq s-c+d \leq$		$c-d$	$\leq$	$c+d$	$\leq$	$c+s-d$

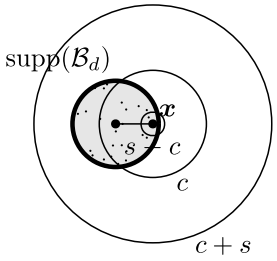
Table 4.1: Cases for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$, where Φ has compact support s. There is an implicit final column with all entries $c+s+d$



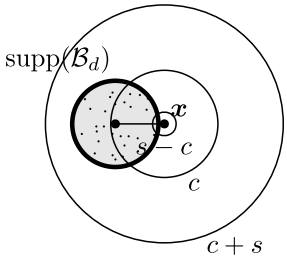
(a) $0 \leq \|\boldsymbol{x}\| < c - d$



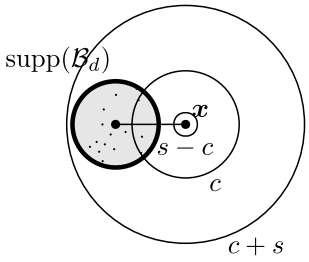
(b) $c - d \leq \|\boldsymbol{x}\| < d - (s - c)$



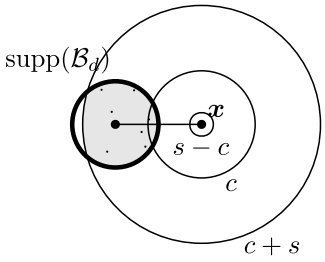
(c) $d - (s - c) \leq \|\boldsymbol{x}\| < d$



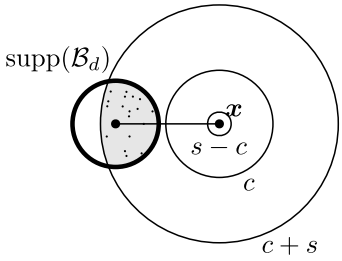
(d) $d \leq \|\boldsymbol{x}\| < s - c + d$



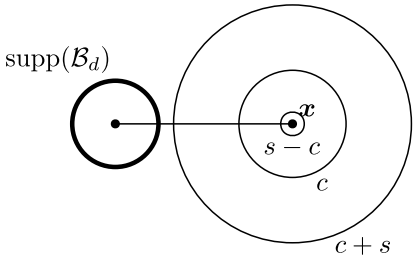
(e) $s - c + d \leq \|\boldsymbol{x}\| < c + s - d$



(f) $c + s - d \leq \|\boldsymbol{x}\| < c + d$

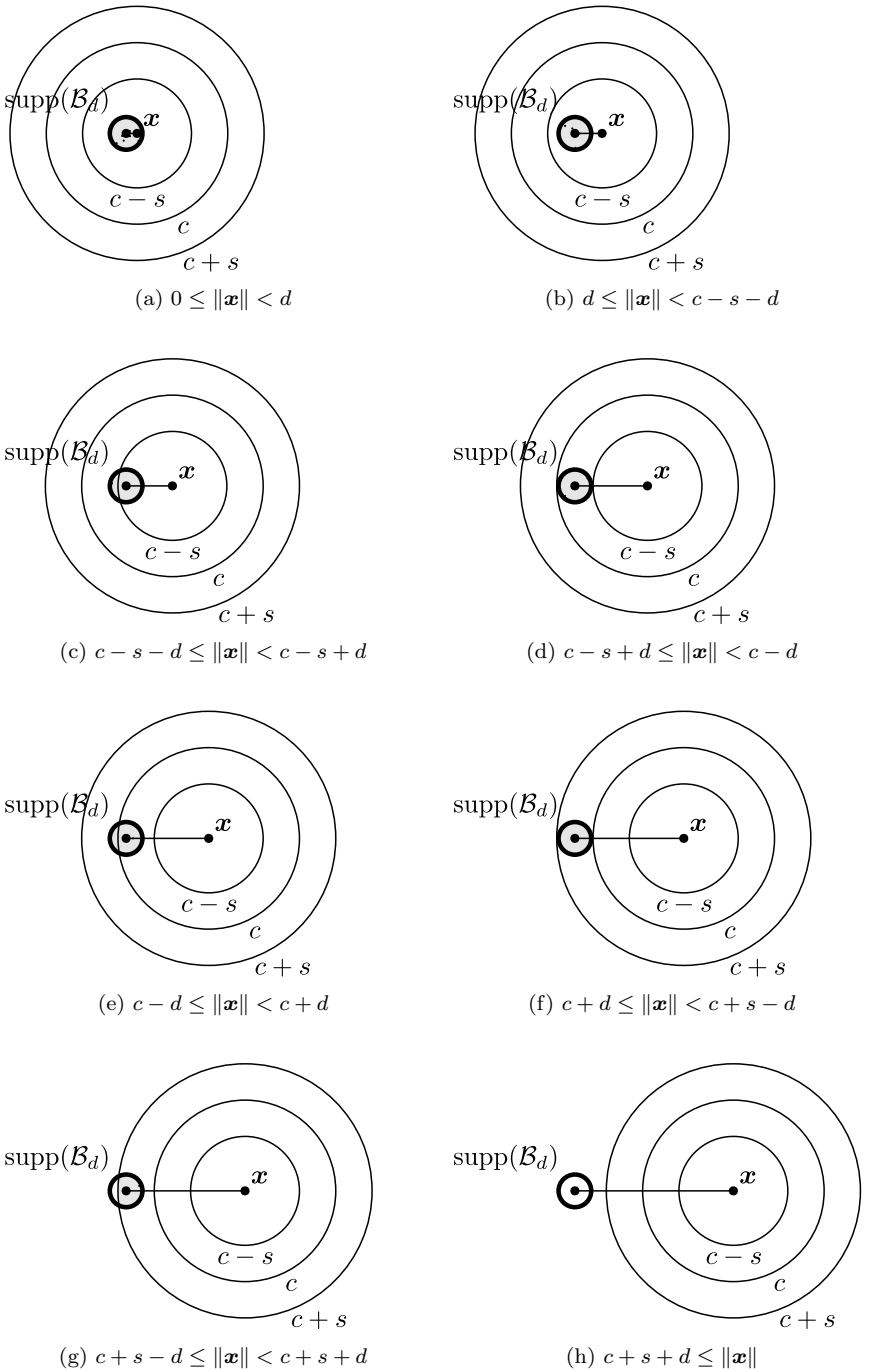


(g) $c + d \leq \|\boldsymbol{x}\| < c + s + d$



(h) $c + s + d \leq \|\boldsymbol{x}\|$

Figure 4.8: Piecewise sections for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$, where Φ has compact support s, and the relative sizes of c and d put us in case F.



(a) $0 \leq \|\boldsymbol{x}\| < d$

(b) $d \leq \|\boldsymbol{x}\| < c-s-d$

(c) $c-s-d \leq \|\boldsymbol{x}\| < c-s+d$

(d) $c-s+d \leq \|\boldsymbol{x}\| < c-d$

(e) $c-d \leq \|\boldsymbol{x}\| < c+d$

(f) $c+d \leq \|\boldsymbol{x}\| < c+s-d$

(g) $c+s-d \leq \|\boldsymbol{x}\| < c+s+d$

(h) $c+s+d \leq \|\boldsymbol{x}\|$

Figure 4.9: Piecewise sections for $\Phi \star \mathcal{B}_c \star \mathcal{B}_d$, where Φ has compact support s, and the relative sizes of c and d put us in case Q.

Chapter 5

Applications

5.1 Track Data

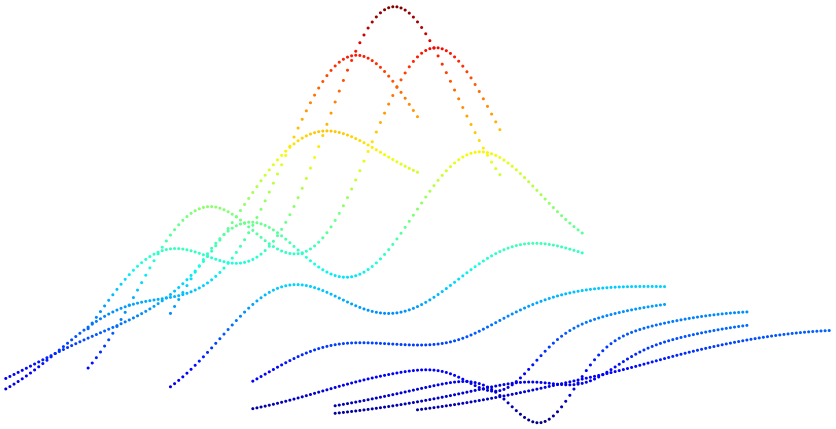
In this section we describe a simple greedy algorithm which uses line sources to approximate a track dataset. The motivation is that the sampling along a track is orders of magnitude denser than the sampling between tracks. It therefore makes little sense to have a point source for every measured point value, so we consider approximating a "segment" of point sources by a single line (segment) source. We will develop a greedy algorithm approach to the fitting task and illustrate it by applying it to a synthetic dataset and to airborne magnetic survey data.

Our synthetic dataset is constructed by sampling Franke's function

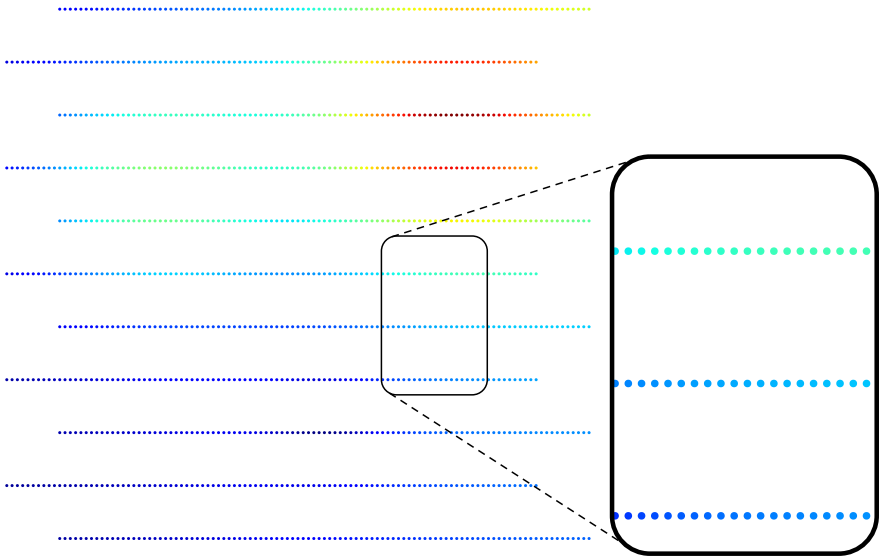
$$\begin{aligned} f(x,y) = \quad & \frac{3}{4}e^{-((9x-2)^2+(9y-2)^2)/4} + \frac{3}{4}e^{-(9x+1)^2/49-(9y+1)/10} \\ & + \frac{1}{2}e^{-((9x-7)^2+(9y-3)^2)/4} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2} \end{aligned} \tag{5.1}$$

along 11 lines parallel to the x-axis and spaced 0.1 units apart from 0 to 1. Each line consists of 101 points 0.009 units apart, for 1111 points in total. To ensure unisolvency (avoid a singular matrix when each line is represented by one line source), the x position of the lines' first points alternates between 0 and 0.1. Finally, as an aid to visualisation, we rotate the entire dataset 180° so that the original origin is in the upper right. This dataset is shown in Figure 5.1.

The real-world datasets we apply our method to are airborne measurements of the Earth's magnetic field over regions of the USA, by the United States Geological Survey [118]. These surveys were conducted over the past several decades by flying near-parallel near-straight tracks and taking point samples along them, sometimes with a few cross tracks perpendicular to the main set. The spacing



(a) Franke's function.



(b) Franke's function tracks.

Figure 5.1: Synthetic track data derived from Franke's function.

and configuration of the point samples and tracks varies considerably between datasets.

From the many datasets available in [118], we choose three in particular to focus on here, MO_1092, CA_3037, and CA_3078. These represent a range of different spacing and configuration possibilities, while remaining small enough to be easily and quickly worked with. Two views of each are shown in Figures 5.2, 5.3 and 5.4. MO_1092 has closely spaced tracks and medium spaced points, CA_3037 has closely spaced tracks and widely spaced points, and CA_3078 has widely spaced tracks and closely spaced points.

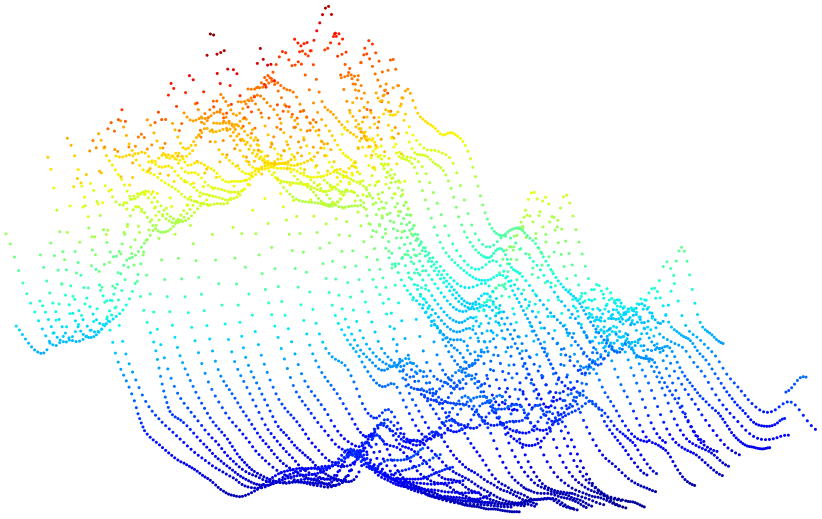
Note from the top down views that the tracks flown by the aircraft are not straight, that not all tracks are sampled at the same spatial frequency, and that the tracks start and stop at different positions, or sometimes cut out only to return later. Also note that the 3D views show little "high frequency" variation along a track. We interpret this as meaning that the data will be well fitted by a smooth surface and that the measurements contain little random noise. Therefore there is no need to use a spline smoothing variant of integral interpolation.

While the tracks themselves can be interpreted as line segments, they are too few and too long to capture the full details of the data when used directly as line sources. However, they can of course be split into multiple segments. In order to derive an adaptive algorithm for splitting the tracks and approximating such datasets with line source RBFs, we need a measurement of error. We define the ℓ_1 error for segment k, e_1^k, of an approximation s to be

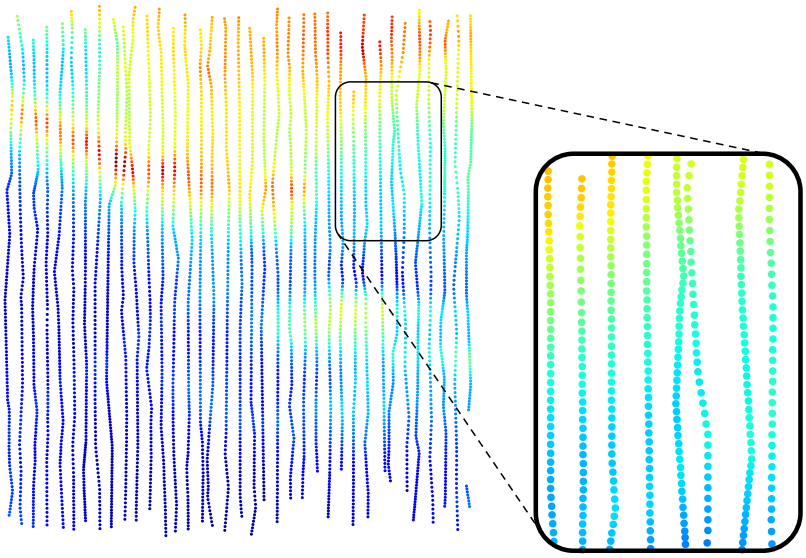
$$e_1^k = \sum_{\boldsymbol{x}_i \in \mathcal{X}_k} |f(\boldsymbol{x}_i) - s(\boldsymbol{x}_i)|\,,$$

where $\mathcal{X}_k$ is the set of all points "on" segment k, and $f(\boldsymbol{x}_i)$ is the data value at $\boldsymbol{x}_i$. With this measure available for comparing the approximation on different segments, we can heuristically choose the segment with the largest error as a good candidate to be subdivided. To determine where the segment should be broken, we find the half error point, the point at which, as we progress along the segment adding up point errors, the cumulative sum of errors reaches half of the total error. We are now ready to define our track data greedy algorithm, shown in Algorithm 2.

The slowest part of this algorithm is the fitting of the integral interpolant, involving numerical integration and the solution of a full linear system at each iteration, neither of which can be easily sped up, but the next slowest part is the calculation of the e_1^ks, which can be. Since there is no guarantee that the line segment with the highest error is the absolute best candidate for subdivision, or that the half error point is the absolute best position to divide it, it is acceptable

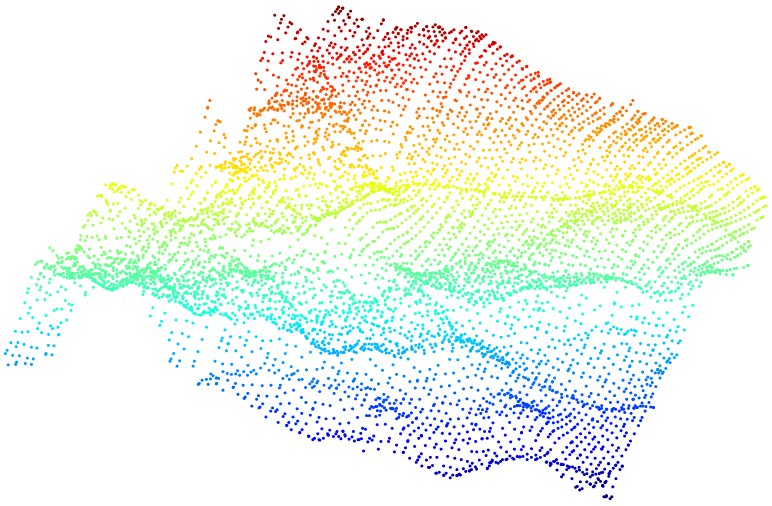


(a) Variation in Earth's magnetic field.

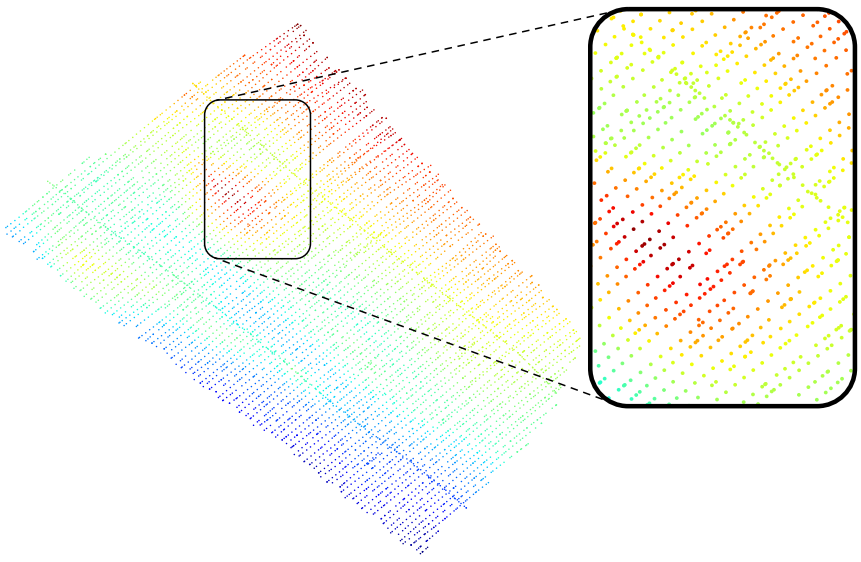


(b) Track variation.

Figure 5.2: Two views of the airborne magnetic survey [118] dataset MO_1092, which contains 38 tracks and 5069 points, and covers an area of approximately 31km×41km in Missouri.

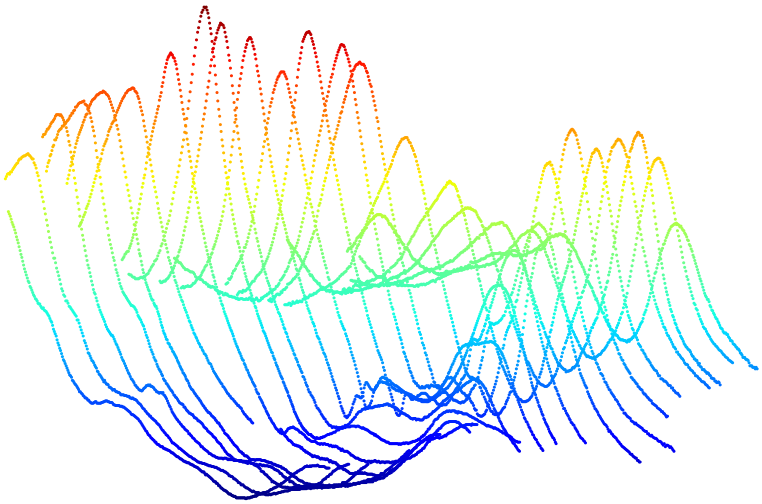


(a) Variation in Earth's magnetic field.

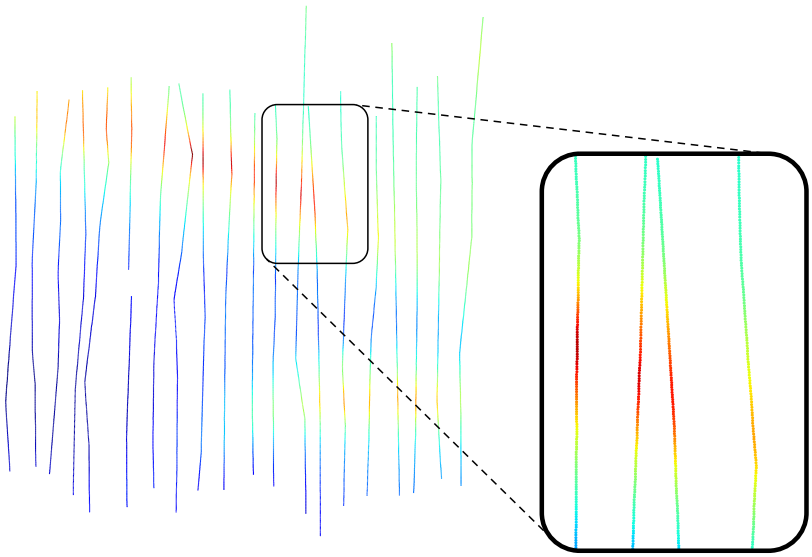


(b) Track variation.

Figure 5.3: Two views of the airborne magnetic survey [118] dataset CA_3037, which contains 98 tracks and 7005 points, and covers an area of approximately 93km×138km in California.



(a) Variation in Earth's magnetic field.



(b) Track variation.

Figure 5.4: Two views of the airborne magnetic survey [118] dataset CA_3078, which contains 21 tracks and 8055 points, and covers an area of approximately 33km×39km in California.

Algorithm 2 Simple Track Data Greedy Algorithm

Input : Set of 2D points $\mathcal{X}$, associated scalar data values $f(\boldsymbol{x}_i)$ for $\boldsymbol{x}_i \in \mathcal{X}$
Output: Line source approximation s to f

1: Divide the points up into tracks, and order them along each track.
2: Make a coarse subdivision of the tracks into line segments, associating each segment with the points it covers and a line source along its length.
3: **repeat**
4: Form a line source approximation s by performing integral interpolation to data averages over segments, using the current list of segments.
5: Calculate the error in the approximation to the subset of data values associated with each segment.
6: Divide a segment associated with the largest error at the half error point, and replace the corresponding line source by two new line sources.
7: **until** satisfied

to take shortcuts which increase speed at the cost of reducing the accuracy with which we find the worst segment and its half error point.

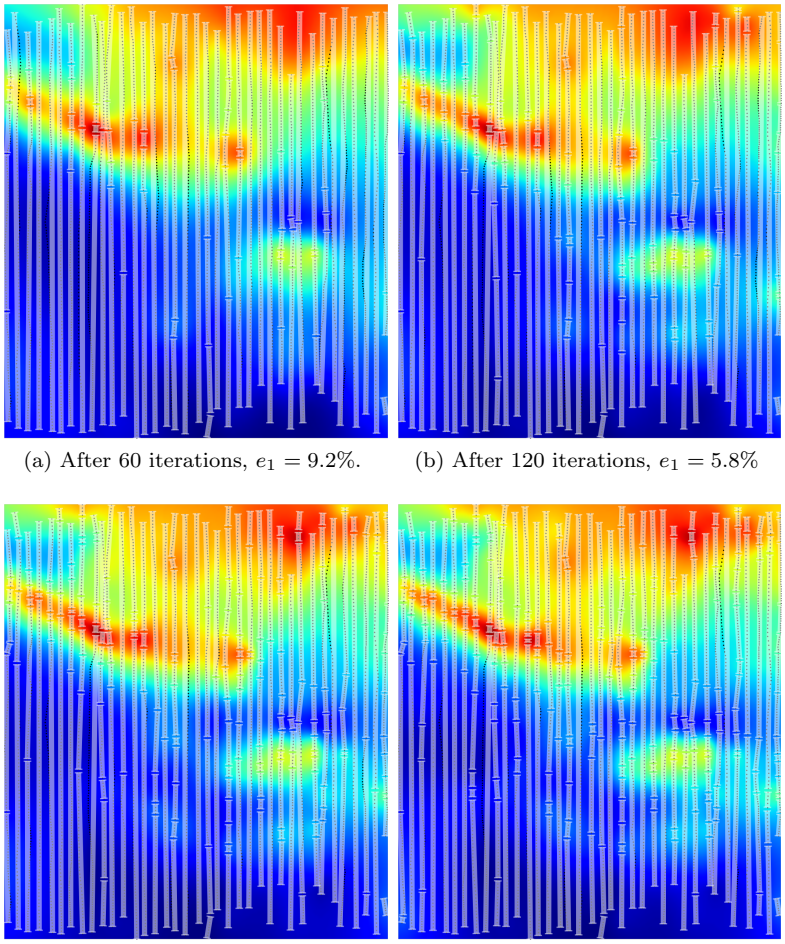
Instead of calculating the full ℓ_1 error, we can calculate a representative part of it by defining an integer parameter n_e and for each line segment selecting n_e points at random, $\mathcal{X}_k^{n_e}$, to contribute to the estimated error, e_e^k:

$$e_e^k = \sum_{\boldsymbol{x}_i \in \mathcal{X}_k^{n_e}} |f(\boldsymbol{x}_i) - s(\boldsymbol{x}_i)| \,.$$

If $\#\mathcal{X}_k \leq n_e$, then $\mathcal{X}_k^{n_e} = \mathcal{X}_k$. The half error point is found in the same way as before, except that the point errors which are summed are only from points in $\mathcal{X}_k^{n_e}$. Using these in Algorithm 2 in place of e_1^k yields similar results faster. In fact, the results with e_e^k are often better than those using e_1^k. Note that e_1^k is contained within e_e^k, since for any n_e at least as large as the maximum number of points in a segment, $e_e^k = e_1^k$. We refer to this case as $n_e = \infty$.

The difficulty preventing the use of matrix updating is that line sources must be removed as well as added. The segments to be removed are very likely to be older ones, in the earlier rows of the matrix, and the natural form of updating involves applying Givens rotation to gradually move the Cholesky factor row to the bottom of the matrix where it can easily be removed. All this extra work is likely to outweigh the benefits of updating, though we have not yet verified this in practice. Another area of future work is the investigation of algorithms which leave old line segments in play while adding smaller refining segments—the old segments are still valid, just not detailed enough—though this seems likely to harm conditioning.

In practice, our initial setup before following Algorithm 2 is to create one line segment per track. For artificial datasets in particular, care must be taken to ensure that the unisolvency (page 9) condition of Theorem 1 (page 10) is



(a) After 60 iterations, $e_1 = 9.2\%$. (b) After 120 iterations, $e_1 = 5.8\%$

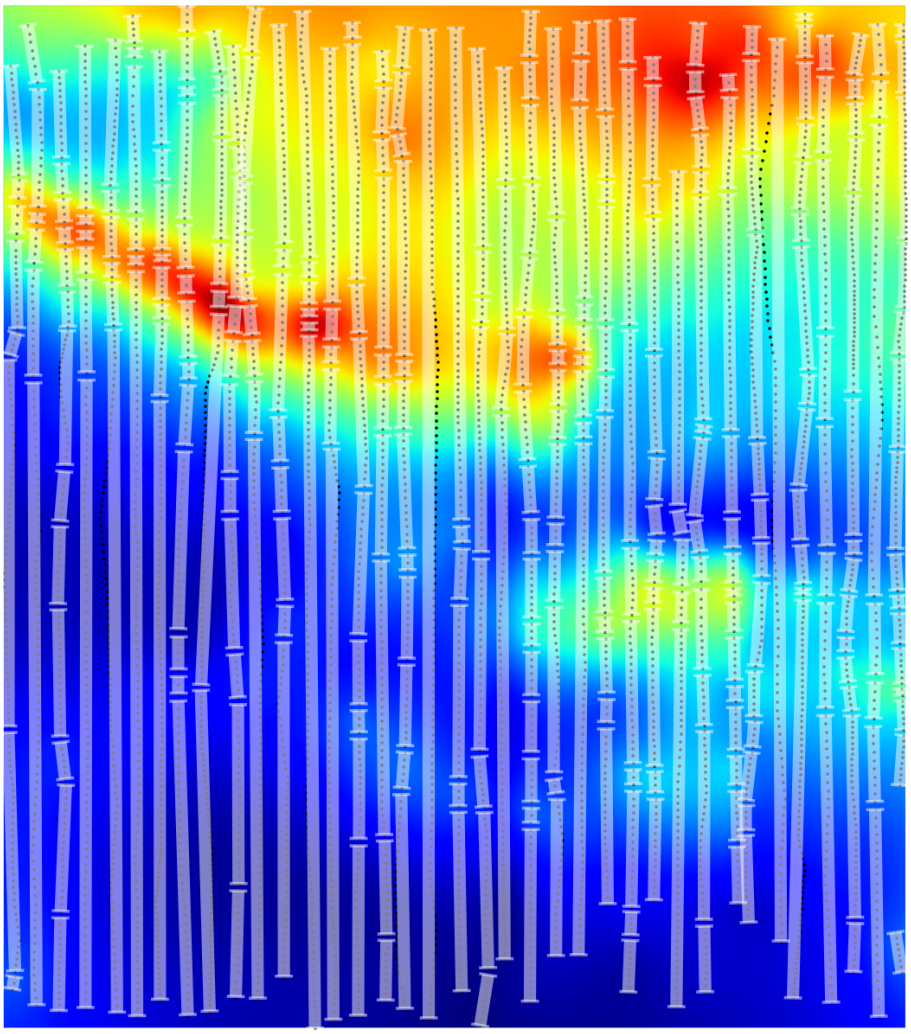
(c) After 180 iterations, $e_1 = 4.2\%$ (d) After 240 iterations, $e_1 = 3.3\%$

Figure 5.5: The greedy algorithm applied to [118] dataset MO_1092, with $n_e = 10$ and using line sources derived from the absolute value basic function.

satisfied. Dividing a single track in half is usually enough to satisfy the conditions for Lemma 3 (page 17) if they were not met already. Our real-world test datasets meet the conditions for Lemma 3 by the nature of their geometry, having some broken or very short tracks which ensure that three tracks can be be chosen which no single line can pass through. Our synthetic dataset does not meet the conditions of Lemma 3, yet is unisolvent by design with half of its tracks offset relative to the others.

In order to compare the results of Algorithm 2 between datasets and different values of n_e, we define the total normalised ℓ_1 error e_1 as

$$e_1 = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{X}} |f(\boldsymbol{x}_i) - s(\boldsymbol{x}_i)|}{\sum_{\boldsymbol{x}_i \in \mathcal{X}} |f(\boldsymbol{x}_i)|} = \frac{\sum_k e_1^k}{\sum_{\boldsymbol{x}_i \in \mathcal{X}} |f(\boldsymbol{x}_i)|},$$



(e) After 300 iterations, $e_1 = 2.6\%$

Figure 5.5: The greedy algorithm's progress continued.

where $\mathcal{X}$ is the set of all points in the dataset.

The progress of the greedy algorithm on the test dataset MO_1092 is illustrated in Figure 5.5. For this example the parent basic function is the linear basic function $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$, and $n_e = 10$. In the figure the black dots are the data points, and the pale straight lines running up the page correspond to the line sources. The colours represent the magnitude of the current fitted function. The line segments go directly from one end point to the other, with the ends marked by small bars. The other points associated with a line segment will, in general, lie close to the segment but not on it. As the algorithm progresses the line segments are divided adaptively by splitting those segments corresponding to the largest error at the approximate half error point. The plots in the figure clearly show the segments being split preferentially where the action is. That is, splits tend to occur where the underlying function varies most rapidly. Visually at least the behaviour of the data has already been completely captured with a 240 line source fit.

The analogous set of calculations were also performed using line sources derived from the thin-plate spline $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^2 \log \|\boldsymbol{x}\|$, and the results, which are not shown, were very similar.

5.1.1 Comparison with Point Sources

The line source approach detailed above has two serious disadvantages which adversely affect its speed in practice. The numerical integration required to set up the linear system slows down the fitting computations dramatically, and any iterative approach involving splitting line segments into smaller ones cannot easily use matrix updating. Pointwise RBF interpolation suffers from neither of these, but does not take the track nature of the data into account.

We define a pointwise greedy algorithm analogous to Algorithm 2 in Algorithm 3. The initialisation steps still deal with tracks in order to start with the same number of sources as in the line source case, and have them arranged roughly similarly. As in that case, we can speed the process up by checking only n_e unused points. Since we can use matrix updating to avoid solving the whole linear system at each iteration (see Section 11.3, page 146 for our method), the evaluations for finding out which point to add next are the most expensive step in the algorithm. Evaluating only n_e points makes for an even bigger speed up than in the line source case, since there it is n_e points per track rather than total, and the evaluations themselves are considerably simpler and faster. In the piecewise case, however, the results for the sped-up version are usually not quite as good.

The progress of the pointwise greedy algorithm on the test dataset MO_1092

Algorithm 3 Simple Pointwise Greedy Algorithm

Input : Set of 2D points $\mathcal{X}$, associated scalar data values $f(\boldsymbol{x}_i)$ for $\boldsymbol{x}_i \in \mathcal{X}$, integer maximum number of points to check n_e

Output: Pointwise approximation s to f

1: Divide the points up into tracks, and order them along each track.
2: Make a point source at the middle point of each track, and form a pointwise approximation s to them.
3: **repeat**
4: Calculate the error in the approximation at at most n_e unused data points.
5: Add a point source at the point with the largest error and update s to include it.
6: **until** satisfied

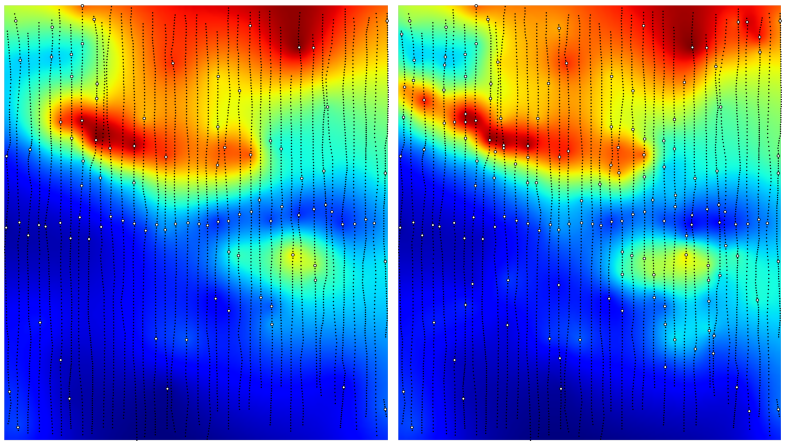
is illustrated in Figure 5.6. For this example the basic function is the linear $\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$, and $n_e = 10$. In the figure the black dots are the data points, and the large white dots are the point sources.

The progress of both algorithms for various values of n_e is shown for each test dataset in Figures 5.7, 5.8, 5.9, and 5.10. For the $n_e < \infty$ cases, the lines plotted are the average of 20 runs, since there is randomness involved. Numeric values for the errors after 150 and 300 iterations are given in Tables 5.1 and 5.2.

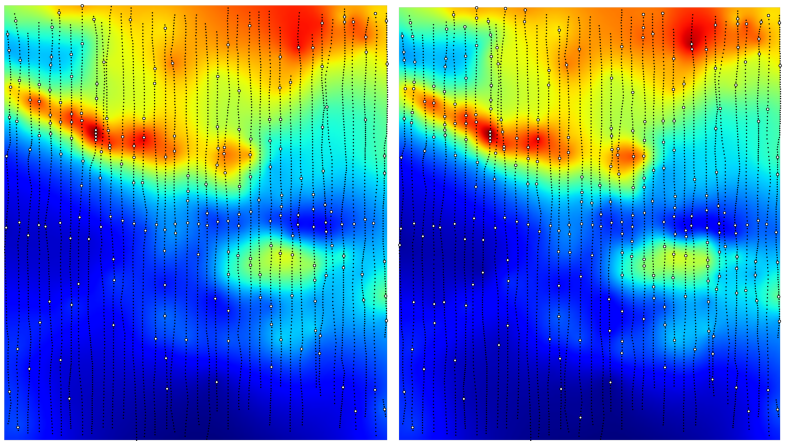
The integral interpolation method runs about two orders of magnitude slower than the pointwise method, though the latter does use more optimised code. Both are implemented in Python [123] using SciPy [55] for vectorised mathematical operations, inline C [58] code and access to LAPACK [3] for solving linear systems and QUADPACK [92] for numerical integration.

As the graphs and tables show, the results depend significantly on the dataset, though perhaps surprisingly there is no obvious correlation with the track nature of the data. Integral interpolation ends up with the lowest error in both the very track-like synthetic dataset and the rather un-track-like CA_3078 dataset. The results for the synthetic dataset have the distinguishing feature that the integral interpolation method produces lower errors early on, though determining the reason for this will require further analysis.

Overall, however, it is the pointwise method which produces consistently good results, especially for the $n_e = 100$ and $n_e = \infty$ versions. This, coupled with the significant difference in speed, make the pointwise method a better choice for practical applications.



(a) After 60 iterations, $e_1 = 9.1\%$. (b) After 120 iterations, $e_1 = 5.9\%$

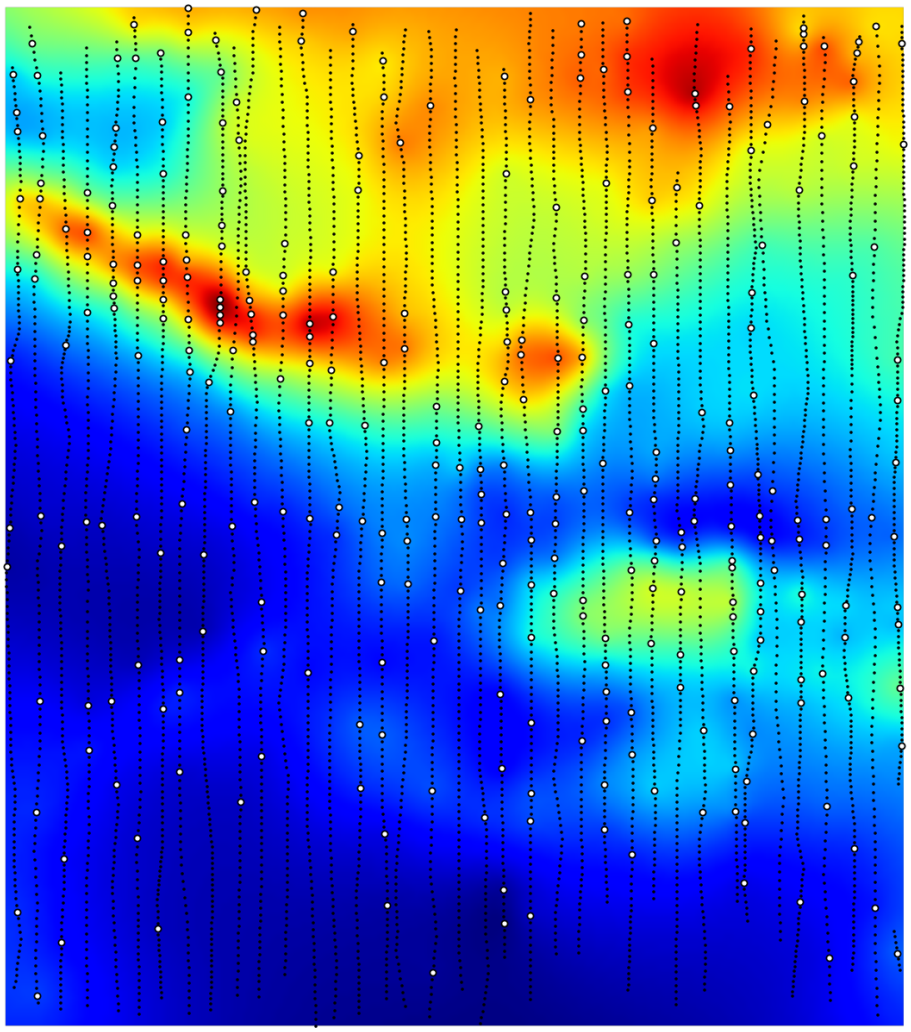


(c) After 180 iterations, $e_1 = 4.5\%$ (d) After 240 iterations, $e_1 = 3.6\%$

Figure 5.6: The pointwise greedy algorithm applied to [118] dataset MO_1092, with $n_e = 10$ and using absolute value point sources.

method	n_e	e_1			
		synthetic	MO_1092	CA_3037	CA_3078
integral	10	0.108%	4.534%	2.191%	2.188%
integral	100	0.104%	6.649%	3.514%	2.229%
integral	∞	0.099%	6.387%	3.401%	2.349%
pointwise	10	0.339%	5.099%	1.550%	2.693%
pointwise	100	0.277%	4.357%	1.333%	2.052%
pointwise	∞	0.258%	3.979%	1.234%	1.752%

Table 5.1: Error measurements for both methods after 150 iterations.



(e) After 300 iterations, $e_1 = 3.0\%$

Figure 5.6: The pointwise greedy algorithm's progress continued.

method	n_e	e_1			
		synthetic	MO_1092	CA_3037	CA_3078
integral	10	0.040%	2.675%	1.443%	0.767%
integral	100	0.039%	3.100%	1.922%	0.726%
integral	∞	0.015%	3.177%	1.870%	0.726%
pointwise	10	0.087%	2.949%	0.935%	1.189%
pointwise	100	0.073%	2.415%	0.823%	0.880%
pointwise	∞	0.073%	2.156%	0.748%	0.718%

Table 5.2: Error measurements for both methods after 300 iterations.

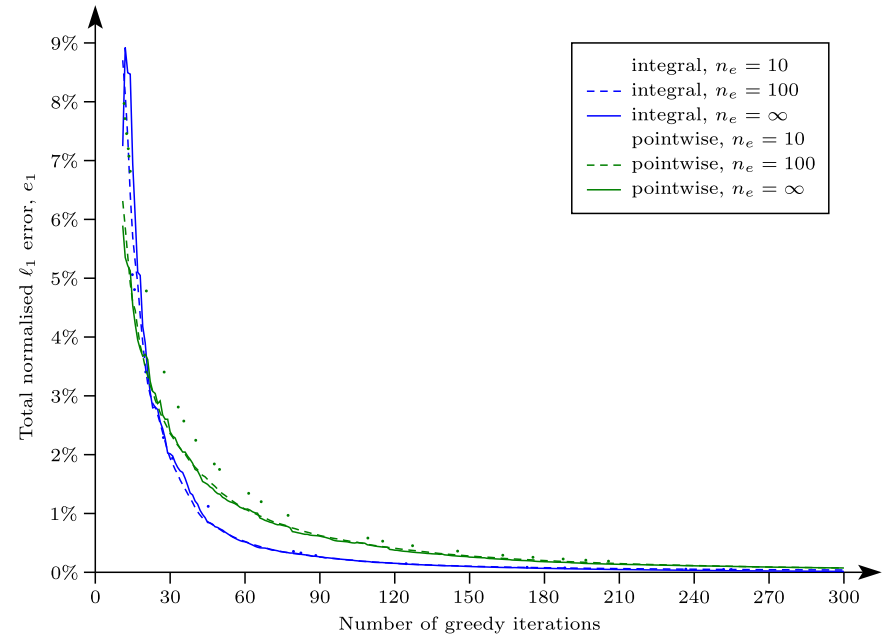


Figure 5.7: Error progress for both methods on the synthetic dataset.

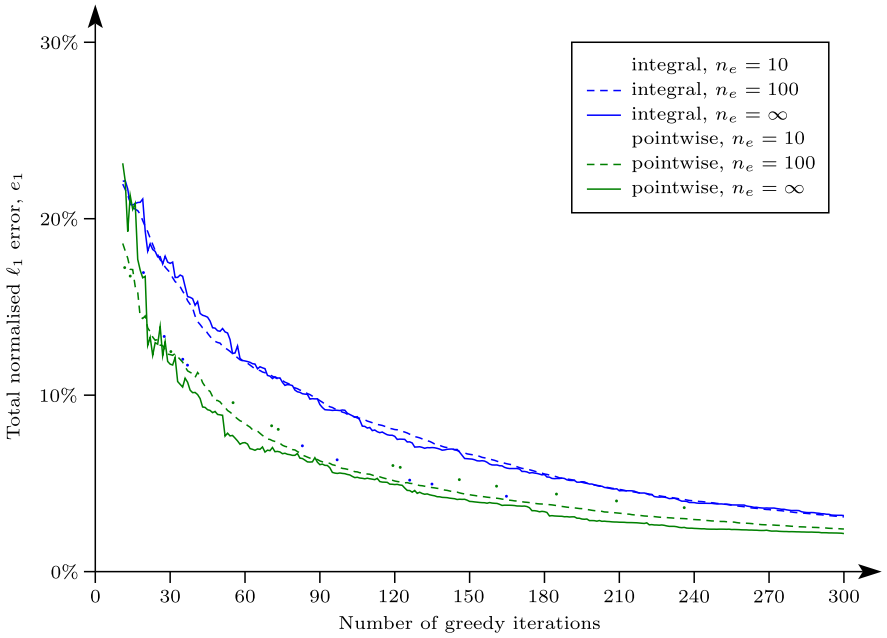


Figure 5.8: Error progress for both methods on the MO_1092 dataset.

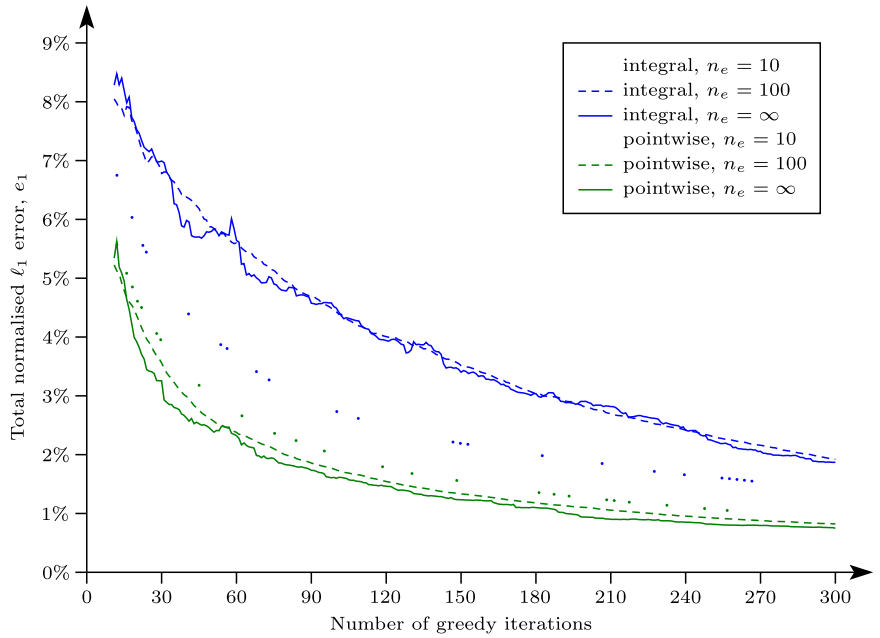


Figure 5.9: Error progress for both methods on the CA_3037 dataset.

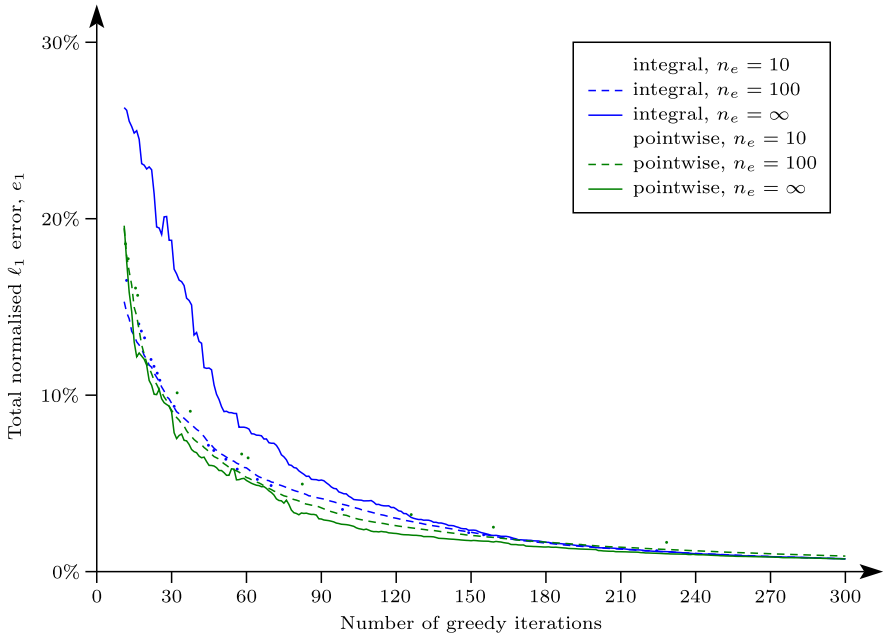


Figure 5.10: Error progress for both methods on the CA_3078 dataset.

5.2 Future Directions

Several other potential areas of application exist for these types of integral interpolation or for the line and ball integrals of Radial Basic Functions.

- Computed Tomography (CT) scans reconstruct a 2- or 3-dimensional density image from a set of X-rays taken from different angles, where each X-ray pixel is a measurement of the line integral of the attenuation coefficients along a ray. This situation is very similar to the track data application described in Section 5.1, but has the appealing features that the data is truly integral in nature, and that the positions of the lines are fixed; many different datasets are reconstructed using different values measured along the same lines. This is convenient for our approach since the numerical quadrature required to find the elements of G (Problem 2, page 10) need only be performed once, and the matrix $Q^T GQ$ (Algorithm 1, page 19) could be factorised or inverted just once, in advance. Important issues to be explored include scaling and how the symmetry of the line source arrangement affects things.

- Density estimation and particular applications of it such as photon mapping will be interesting uses for our ball integral interpolation.

- Our ball source formulas could also be used for data smoothing via the implicit smoothing technique of Beatson and Bui, 2003 [8]. In that technique one first interpolates to noisy data using the basic function Φ, and then on evaluation replaces Φ by the smoother function $\Psi = \Phi \star K$. The formulas developed in Section 4.2 are Ψ for various choices of parent Φ, when K is chosen as the normalised characteristic function of a sphere with radius c. This is equivalent to finding the moving average of Φ with a ball as the "window".

- Further theoretical advancements may be possible, such as an unsymmetric formulation analogous to the Hermite form used for solving differential equations introduced by Kansa in 1990 [57], and analysed theoretically by Hon and Schaback [46] and Schaback [101]. The unsymmetric integral interpolation system seems to be solvable most of the time (though by analogy to other situations, probably not always), but there is as yet no fully developed theory to back this up. Also, it may be possible to find more analytic formulas for integral sources, such as the line source double integrals, ball source integrals in even dimensions, or for other shapes such as rectangles.

Part II

Piecewise Surface Reconstruction

Chapter 6

Introduction

In Part II we turn our attention to the problem of efficiently fitting an approximation to a large scattered dataset. While the techniques we develop here are general and broadly applicable, we focus, in particular, on the problem of reconstructing a continuous surface in 3-space from an unorganised set of discrete points.

6.1 Motivation and Existing Solutions

In this application we have a collection (cloud) of points, for which we know only their locations in space, the fact that they are on the unknown surface, and their surface normal (possibly estimated from point geometry). We assume that this surface is manifold—that it could exist in the real world as the surface of a solid object. Most datasets that we are interested in solving this problem for are derived from real objects (e.g. by laser scan).

Surface reconstruction from point cloud data is an important problem with numerous applications. For example, point cloud data sets are generated from laser scans for such wide-ranging applications as rapid prototyping, simulation, prosthetics, custom replacement parts, archaeology and visual effects. These point clouds are inherently scattered, and holes arise naturally as often the scanner cannot see certain parts of the surface. A well established class of techniques models the unknown surface implicitly as the set of points X where a function $s(\boldsymbol{x})$ is zero. This tends to be superior to the alternative parametric approach when the sampling density varies widely, when the scan includes holes, and when the surface bifurcates in some unpredictable manner.

The problem now becomes one of finding such a function $s(\boldsymbol{x})$ which is zero at each point in X, positive "inside" them and negative "outside" them. When we add some number of extra points to clarify the inside-outside question, giving

them a value of plus or minus their distance from the nearest surface point, we are left with a scattered interpolation problem well suited to Radial Basis Functions (RBFs).

Fitting an RBF interpolant, however, appears at first glance to involve solving a matrix system which takes $O(n^3)$ operations. This is clearly a problem given that datasets of millions of points are not uncommon, and researchers have taken a number of different approaches towards solving it. These include using only part of the data, by thinning (see e.g. [52] for a general treatment) or fitting greedily (e.g. [16]); dealing with only part of the data at a time, by using domain decomposition (e.g. [10]) or compactly supported basic functions (e.g. [76]); and, most recently, combining large numbers of small, independent fits with a partition of unity.

This last, despite having its groundwork laid decades ago, has appeared only surprisingly recently. Many authors have suggested using local polynomial fits to scattered data, and a survey can be found in [6]. In particular, in 1977, Franke [31] described a generalisation of a 1973 method of Maude's [70] which uses partition of unity to combine local quadratic (or other) approximations which interpolate in circular (or other shaped) subdomains, and in 1980 Franke and Nielson [33, Method I] described a modification of Shepard's [105] 1968 method using partition of unity to combine local quadratic approximations based on points in circular subdomains. These methods are generally formulated and discussed in two dimensions, but can be made to work in any dimension. Maude [70] and Renka [95] (following [33]) let the approximation influence radius differ between points, defining it as the radius of a ball just large enough to contain a certain fixed number of points. Approaches based on these methods have been applied in many areas, for example for interpolation in meteorology [117] and for deformation in geometric modelling [98] and biology [91]. Excepting the survey [6] and the general interpretation of [31], all the methods in this paragraph can be considered "modified" Shepard's methods, where each point has an associated approximation (local or global), and a weight function that decreases radially. As researchers have dealt with larger datasets, locality has become important, and as the local approximations have been made more sophisticated, the number of subdomains has usually been reduced, leading away from modified Shepard to general partition of unity.

In 1982, Franke [32] introduced Radial Basis Functions as the local approximations, using thin plate splines in rectangular subdomains in two dimensions. This approach saw limited use until 2002, when RBFs were used as the local approximations by Lazzaro and Montefusco [65], in a modified Shepard's method, and by Wendland [128], using general partition of unity.

In 2003, researchers began to apply methods along these lines to surface re-

construction. Ohtake et al [83] presented a surface reconstruction algorithm based on a multilevel partition of unity combination of local quadratic approximations in spherical subdomains arranged in a hierarchical grid. Ohtake, Belyaev, and Seidel [85, 84] presented a modified version of [83] in which the subdomains are themselves scattered, forming a modified Shepard's method at the final level with one for each point, and which adds a refining compactly supported "normalised" RBF to each level that interpolates at the centres of the subdomains. Xie et al [136] reconstructed surfaces from noisy point clouds with a modified Shepard's method using local quadrics.

In 2004, Ohtake, Belyaev, and Seidel [86] presented a single level version of [85, 84] which uses an adaptive sphere covering and a least squares RBF or ridge regression approach to approximate rather than interpolate. They also noted that if one only cares about the 0-isosurface then a full partition of unity is unnecessary and the local approximations, which have similar zero level sets, can simply be added together. Also in 2004, Tobor, Reuter and Schlick [114, 115], and Pouderoux, Gonzato, Tobor, and Guitton [93] presented algorithms based on partitions of unity of RBF approximations in ellipsoidal and box-shaped subdomains arranged on hierarchical grids, the latter publication focusing on the 2-dimensional case of digital elevation models.

In 2005, Ohtake, Belyaev, and Seidel [87] continued their approach of [85, 84], using the addition method of [86] rather than partition of unity. Wu, Wang, and Xia [132] presented a method using partitions of unity of RBFs in boxes on a hierarchical grid, very similar to that of Tobor, Reuter and Schlick, but with a way to reduce the number of off-surface points and just a single level. Casciola et al [17] applied the method of [65] to the surface reconstruction problem.

In 2006, Ohtake, Belyaev, and Seidel [89] re-presented their approach of [86]. Tobor, Reuter and Schlick [116] presented a more comprehensive article on their method of 2004. Xia, Wang and Wu [135] used Orthogonal Least Squares to thin the point set, and changed their subdomains to spheres. In 2007, Chen and Lai [18] presented another variation on the theme of RBFs in a hierarchical grid of boxes, this time with a greedy algorithm and matrix updating.

Here, we develop a partition of unity method based on (Hermite) Radial Basis Function approximations in scattered spherical data-driven subdomains, avoiding dependencies of the subdomains on the coordinate system or the approximation itself, avoiding the need for any large single approximation, even a sparse one, and yet still providing good hole filling and scaling to very large datasets and approximations.

6.2 Outline

In Chapter 7, we describe the different parts of our method: multilevel interpolation, partition of unity and how it works with Hermite approximations, Hermite RBFs, sphere coverings to generate subdomains, and our two methods for blending down to coarser levels for hole filling, late hole detection and level-aware partition of unity. We also discuss present and future approaches for scaling to very large datasets.

In Chapter 8, we present results of our sphere covering method, and surface reconstructions comparing levels, hole filling techniques, and Hermite and pointwise approximations. We also compare our method to that of Ohtake, Belyaev, and Seidel [85, 84], and investigate the scaling behaviour of our approach.

Chapter 7

Sphere-based Piecewise RBFs

7.1 Multilevel Implicit Surfaces

Hierarchical approximation schemes are well established as a multiresolution technique, and are particularly useful when the approximation at each level is local in nature. See for example Floater and Iske [29], and Iske's in-depth monograph [53]. The concept can be viewed as an iterative refinement procedure for approximating a function f.

The approximation is constructed with a set of levels. Associated with each level are an approximation, a residual, and a set of data points. The sets of data points are nested, with the set at level L being a superset of that corresponding to level $\mathrm{L}-1$.

An initial, probably quite inaccurate, approximation σ_0 to f is formed, and the corresponding residual is defined as $r_1 = f - \sigma_0$. After levels 0 through $\mathrm{L}-1$ have been processed the current approximation is $\sigma_0 + \cdots + \sigma_{\mathrm{L}-1}$ and the current residual is $r_{\mathrm{L}} = f - (\sigma_0 + \cdots + \sigma_{\mathrm{L}-1})$. The work at level L is to form an approximation σ_{L} to this residual.

We adopt the convention of calling the coarsest level, when $\mathrm{L} = 0$, the *base* level, and saying that the other levels are layered on successively *above* each other.

In the current work, the approximations σ_{L} for $\mathrm{L} > 0$ are constructed via a partition of unity mechanism, using spherical subdomains. Thus the approximations associated with the levels consist of sets of spheres $\{\mathcal{S}_i\}$, weight functions $\{w_i\}$ with $\mathrm{supp}(w_i) = \mathcal{S}_i$, and local approximations $\{s_i\}$. These methods will be developed in the next few sections.

Our base level approximation σ_0 is a globally supported (Hermite) RBF fit, so that the complete multilevel approximation will be defined everywhere. Since σ_0 will be evaluated a great many times (as part of every multilevel evaluation, including all residual calculations during fitting) it is important that it only involve a small number of RBF centres, such as a few hundred. In order to do a reasonable job with a complex model and so few centres, we use a greedy algorithm, Algorithm 4 in Section 7.5. Efficiency of evaluation is more important than the efficiency of the fit here, as for anything but small models, the time taken in fitting the base level is very small compared to the total multilevel fit time.

7.2 (Hermite) Partition of Unity

Following standard partition of unity techniques, we assign to each sphere $\mathcal{S}_i$ a nonnegative radial weight function $w_i(\boldsymbol{x})$ with $\mathrm{supp}(w_i) = \mathcal{S}_i$, which goes smoothly to zero at the boundary $\partial\mathcal{S}_i$. The usual partition of unity blended approximation is

$$\sigma(\boldsymbol{x}) = \sigma_{\mathrm{L}}(\boldsymbol{x}) = \frac{\sum_i w_i(\boldsymbol{x}) s_i(\boldsymbol{x})}{\sum_j w_j(\boldsymbol{x})} = \sum_i v_i(\boldsymbol{x}) s_i(\boldsymbol{x}), \tag{7.1}$$

where $v_i(\boldsymbol{x}) = \frac{w_i(\boldsymbol{x})}{\sum_j w_j(\boldsymbol{x})}$, for $\boldsymbol{x}$ in the interior of $\cup \mathcal{S}_i$. From (7.1), since $\sum_i v_i(\boldsymbol{y}) = 1$, we can satisfy the interpolation condition $\sigma(\boldsymbol{y}) = r(\boldsymbol{y})$, at data points $\boldsymbol{y}$, by requiring $s_i(\boldsymbol{y}) = r(\boldsymbol{y})$ for each i such that $\boldsymbol{y} \in \mathcal{S}_i$. That is, each local approximation interpolates to the data points in its subdomain, and hence they agree in regions of overlap.

Also, if at a data point $\boldsymbol{y}$ we know the directional derivative $(D_{\boldsymbol{u}} r)(\boldsymbol{y})$, then we can follow the above idea and set both $s_i(\boldsymbol{y}) = r(\boldsymbol{y})$ and $(D_{\boldsymbol{u}} s_i)(\boldsymbol{y}) = (D_{\boldsymbol{u}} r)(\boldsymbol{y})$, for each i such that $\boldsymbol{y} \in \mathcal{S}_i$. Then (7.1) implies

$$\begin{aligned}
(D_{\boldsymbol{u}}\sigma)(\boldsymbol{y}) &= \sum_i v_i(\boldsymbol{y}) (D_{\boldsymbol{u}} s_i)(\boldsymbol{y}) + \sum_i s_i(\boldsymbol{y}) (D_{\boldsymbol{u}} v_i)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} r)(\boldsymbol{y}) \sum_i v_i(\boldsymbol{y}) + r(\boldsymbol{y}) \sum_i (D_{\boldsymbol{u}} v_i)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} r)(\boldsymbol{y}) \sum_i v_i(\boldsymbol{y}) + r(\boldsymbol{y}) \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} r)(\boldsymbol{y}),
\end{aligned}$$

since $\sum_i v_i(\boldsymbol{y}) = 1$. So, unsurprisingly, the partition of unity will satisfy Hermite interpolation conditions when each local approximation satisfies the conditions for all data points in its subdomain.

For our weight function we use the bell-shaped C^2 Wendland $\Phi_{3,1}$:

$$w_i(\boldsymbol{x}) = \left(1 - \frac{\|\boldsymbol{x} - \boldsymbol{c}_i\|}{r_i}\right)_+^4 \left(4\frac{\|\boldsymbol{x} - \boldsymbol{c}_i\|}{r_i} + 1\right),$$

where $\boldsymbol{c}_i$ and r_i are the centre and radius of sphere $\mathcal{S}_i$, respectively, and the $(\cdot)_+$ notation, as in Part I, is defined as

$$(a)_+ = \begin{cases} 0, & a \leq 0 \\ a, & a > 0. \end{cases}$$

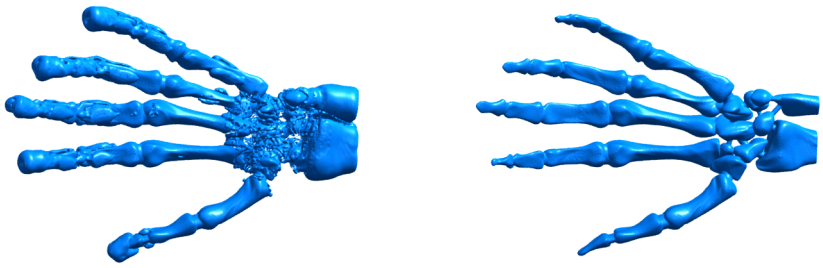
At a high level L the spheres are small, making $\sigma(\boldsymbol{x})$ undefined except in a narrow shell $\cup\mathcal{S}_i$, containing the data points. However, we have approximations at lower levels which are defined over larger regions, and ultimately a global approximation defined everywhere. It is therefore natural to supplement the definition of $\sigma(\boldsymbol{x})$ given by (7.1) by specifying $\sigma(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \notin \cup\mathcal{S}_i$, making multilevel blending straightforward.

Unfortunately, if the local approximation $s_i(\boldsymbol{x})$ does not go to zero as $\boldsymbol{x}$ approaches the boundary $\partial\mathcal{S}_i$, this formulation will leave us with a discontinuous transition to zero at points on the boundary of a single sphere $\mathcal{S}_i$ which are not covered by other spheres. We therefore need a way of blending down to lower levels, and we have developed two methods, described in Sections 7.6 and 7.7, which also solve the problem of holes between spheres, as described in Section 7.5.

7.3 Small Local Approximations

Within each spherical subdomain $\mathcal{S}_i$ we fit an approximation $s_i(\boldsymbol{x})$ to the residual of the previous level r_L. There is great scope for variation in the choice of approximant, and as these subdomains are entirely independent of each other, the approximation method can even be chosen on a per-sphere basis. In practice we use either a pointwise or Hermite RBF fit.

While the points inside a particular sphere are clearly the most important for the local approximation, including points from outside is also perfectly valid. Including "mid-range" points, such as the centres of neighbouring spheres, can improve the extrapolatory behaviour of the local approximation and thus the final blended result.



(a) Pointwise approximation with incorrectly positioned off-surface points.

(b) Hermite approximation.

Figure 7.1: Possible effects of poor off-surface distance selection.

7.4 Hermite RBFs

In this section we consider Hermite interpolation by Radial Basis Functions. Work on this problem is scattered in the literature, with solvability/poisedness of interpolation problems shown under a variety of assumptions and in a variety of often quite abstract settings. For many applications, including surface reconstruction, full generality is not needed. Therefore we keep this section directly relevant to our application and accessible by concentrating on a special case involving interpolation to function values and directional derivatives, and using the language of calculus rather than that of distributions.

We have two motivations for using Hermite RBFs for this application. Firstly, the number of interpolation conditions is reduced, for example from two off-surface points per surface point to one directional derivative per surface point, and secondly, the possibility of choosing incorrect off-surface distances is removed (see Figure 7.1 for an example of this problem[1]).

Amongst the literature the early work of Duchon [23] contains a general theory which can be applied to Hermite interpolation by polyharmonic splines and pseudo splines. Wu [133] and Sun [110] cover other choices of basic function and specifications of interpolation conditions. Iske [51] has a very general setting allowing great flexibility in the choice of functionals and basic function.

We will consider point evaluations, directional derivatives and iterated directional derivative evaluations, treating the first two evaluation types as subsets of the third. An iterated directional derivative evaluation of the form

[1] Skeleton hand model courtesy of the Clemson University Stereolithography Archive.

$\eta_i(f) = \left(D_{\boldsymbol{u}_{i,h}} \dots D_{\boldsymbol{u}_{i,2}} D_{\boldsymbol{u}_{i,1}} f\right)(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\{\boldsymbol{u}_{i,j}\}$ are unit vectors in $\mathbb{R}^d$, will be called a directional derivative functional of order h. A directional derivative of the form $\eta_i(f) = (D_{\boldsymbol{u}_i} f)(\boldsymbol{x}_i) = \nabla f(\boldsymbol{x}_i) \cdot \boldsymbol{u}_i$, where $\boldsymbol{u}_i$ is a unit vector in $\mathbb{R}^d$, will be called a directional derivative functional of order 1, and a point evaluation of the form $\eta_i(f) = f(\boldsymbol{x}_i)$ will be called a directional derivative functional of order 0. The space of all directional derivative functionals of order κ or less will be denoted $\mathcal{M}_\kappa$. We consider Hermite interpolation problems of the following form:

Problem 3 (Hermite interpolation)**.** *Given directional derivative functionals $\eta_1, \dots, \eta_m \in \mathcal{M}_\kappa$ and corresponding values $b_1, \dots, b_m$, find a function $s \in C^\kappa(\mathbb{R}^d)$ such that*

$$\eta_i(s) = b_i, \quad 1 \le i \le m. \tag{7.2}$$

The problem has solutions $s \in C^\kappa(\mathbb{R}^d)$ provided the m functionals are linearly independent over $C^\kappa(\mathbb{R}^d)$. As in Part I, we write $\pi^d_{k-1} = \pi^d_{k-1}(\mathbb{R}^d)$ for the space of polynomials of total degree not exceeding $k-1$ in d variables. Then the symmetric formulation of the Hermite RBF interpolation problem is as follows:

Problem 4 (Hermite RBF interpolation)**.** *Given a basic function $\Phi \in C^{2\kappa}(\mathbb{R}^d)$ find a function of the form*

$$s(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{i=1}^{m} c_i \eta_i^{\boldsymbol{y}} \Phi(\boldsymbol{x} - \boldsymbol{y}), \quad p \in \pi^d_{k-1}(\mathbb{R}^d), \tag{7.3}$$

satisfying the interpolation conditions (7.2) *and the side conditions*

$$\sum_{i=1}^{m} c_i \eta_i(q) = 0, \text{ for all } q \in \pi^d_{k-1}. \tag{7.4}$$

The $\boldsymbol{y}$ superscript on the functionals in (7.3) *indicates that they are applied with respect to the $\boldsymbol{y}$ variable.*

We now give definitions of conditional positive definiteness and unisolvency appropriate for the Hermite RBF interpolation problem, Problem 4.

Definition 4. *A $C^{2\kappa}(\mathbb{R}^d)$ function Φ is* strictly conditionally positive definite *of order k, smoothness κ on $\mathbb{R}^d$, written $\Phi \in \mathsf{HSPD}_{k,\kappa}(\mathbb{R}^d)$, if given any finite linearly independent set of functionals $\{\eta_1, \dots, \eta_m\} \subset \mathcal{M}_\kappa$ the $m \times m$ matrix G, with $G_{ij} = \eta_i^{\boldsymbol{x}} \eta_j^{\boldsymbol{y}} \Phi(\boldsymbol{x} - \boldsymbol{y})$, satisfies*

$$\lambda^T G \lambda > 0,$$

whenever both $\boldsymbol{c} \neq \mathbf{0}$ *and*

$$\sum_{i=1}^{m} c_i \eta_i(q) = 0, \qquad \text{for all } q \in \pi_{k-1}^d.$$

Definition 5. *A set of functionals* $\{\eta_1, \ldots, \eta_m\} \subset \mathcal{M}_\kappa$ *will be called* unisolvent *for* π_{k-1}^d *if the only polynomial* $p \in \pi_{k-1}^d$ *for which all the functionals are zero is the zero polynomial.*

An argument along familiar RBF theory lines gives us the following theorem:

Theorem 3. *If* $\Phi \in \mathsf{HSPD}_{k,\kappa}$ *and the functionals* $\eta_1, \ldots, \eta_m \subset \mathcal{M}_\kappa$ *are both linearly independent over* $C^\kappa(\mathbb{R}^d)$ *and unisolvent for* π_{k-1}^d*, then the Hermite RBF interpolation problem, Problem 4, has a unique solution. As in the integral interpolation case in Section 3.1, let* $\ell = \dim\left(\pi_{k-1}^d\right)$ *and let* $\{p_1, \ldots, p_\ell\}$ *be a basis of* π_{k-1}^d*. Then the coefficients of this solution may be found by solving the linear system arising from Equations* (7.3) *and* (7.4)*:*

$$\begin{bmatrix} G & P \\ P^T & O \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \mathbf{0} \end{bmatrix}, \tag{7.5}$$

where G *is* $m \times m$ *with* $G_{ij} = \eta_i^{\boldsymbol{x}} \eta_j^{\boldsymbol{y}} \Phi(\boldsymbol{x} - \boldsymbol{y})$*,* P *is* $m \times \ell$ *with* $P_{ij} = \eta_i p_j$*, and* $p = \sum_{j=1}^{\ell} a_j p_j$*.*

The proof is very similar to the proof of the analogous Theorem 1 in Part I.

Proof of Theorem 3. Consider the case when the right hand side of the linear system (7.5) is zero. Again following well known arguments from the pointwise positive definite case, multiply the first row of the block system (7.5) on the left by $\boldsymbol{c}^T$. This yields

$$\mathbf{0} = \boldsymbol{c}^T G \boldsymbol{c} + \boldsymbol{c}^T P \boldsymbol{a} = \boldsymbol{c}^T G \boldsymbol{c} \quad \text{since } P^T \boldsymbol{c} = \mathbf{0}.$$

Since Φ is strictly conditionally positive definite, this implies $\boldsymbol{c} = \mathbf{0}$, and substituting back, the first row of the block system becomes $P\boldsymbol{a} = \mathbf{0}$. But $P\boldsymbol{a}$ is a vector whose i-th component is η_i applied to the polynomial $q = \sum_{j=1}^{\ell} a_j p_j$, and so the unisolvency of $\{\eta_i\}$ implies $\boldsymbol{a} = 0$. Therefore the only solution to the homogeneous equation is the trivial one and the matrix on the left of equation (7.5) is invertible. Hence, there is a unique solution for any given right hand side. □

We now restrict discussion to the special case of Hermite interpolation problems involving only point evaluations and non iterated directional derivatives, which is of particular interest for surface reconstruction.

It will be convenient to order and label the functionals by type so that $\eta_1, \ldots, \eta_n$ are point evaluation functionals $\eta_i(f) = f(\boldsymbol{x}_i)$ for $1 \le i \le n$ and $\eta_{n+i}(f) = D_{\boldsymbol{u}_i} f(\boldsymbol{y}_i)$, for $1 \le i \le m-n$ are directional derivatives. Correspondingly the values to interpolate will be split into function values $f_i = b_i,\ 1 \le i \le n$ and directional derivative values $g_i = b_{n+i},\ 1 \le i \le m-n$. Note that there is no requirement here that the points $\{\boldsymbol{x}_i\}$ be distinct from the points $\{\boldsymbol{y}_i\}$—in particular, the formulation does allow specifying both function value and gradient at some or all points. Then the solution s takes the form

$$s(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{i=1}^{n} d_i \Phi(\boldsymbol{x} - \boldsymbol{x}_i) + \sum_{i=1}^{m-n} e_i \left(D_{\boldsymbol{u}_i} \Phi\right)(\boldsymbol{y}_i - \boldsymbol{x})$$

where $p = \sum_{i=1}^{\ell} a_i p_i \in \pi_{k-1}^d$ and $\boldsymbol{c} = \left[\begin{smallmatrix} \boldsymbol{d} \\ \boldsymbol{e} \end{smallmatrix}\right]$. Problem 4 can be rewritten as the linear system to solve for parameters $\boldsymbol{d}$, $\boldsymbol{e}$, and $\boldsymbol{a}$

$$\begin{bmatrix} A & B^T & Q \\ B & C & R \\ Q^T & R^T & O \end{bmatrix} \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{e} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix}. \tag{7.6}$$

Here,

$$\begin{array}{lll}
A_{n\times n} & \text{has } A_{ij} = & \eta_i^{\boldsymbol{x}} \eta_j^{\boldsymbol{y}} \Phi(\boldsymbol{x}-\boldsymbol{y}) = \Phi(\boldsymbol{x}_i - \boldsymbol{x}_j), \\
B_{(m-n)\times n} & \text{has } B_{ij} = & \eta_{n+i}^{\boldsymbol{x}} \eta_j^{\boldsymbol{y}} \Phi(\boldsymbol{x}-\boldsymbol{y}) = \left(D_{\boldsymbol{u}_i} \Phi\right)(\boldsymbol{y}_i - \boldsymbol{x}_j), \\
C_{(m-n)\times(m-n)} & \text{has } C_{ij} = & \eta_{n+i}^{\boldsymbol{x}} \eta_{n+j}^{\boldsymbol{y}} \Phi(\boldsymbol{x}-\boldsymbol{y}) = -\left(D_{\boldsymbol{u}_i} D_{\boldsymbol{u}_j} \Phi\right)(\boldsymbol{y}_i - \boldsymbol{y}_j), \\
Q_{n\times \ell} & \text{has } Q_{ij} = & \eta_i p_j(\boldsymbol{x}) = p_j(\boldsymbol{x}_i), \\
R_{n\times \ell} & \text{has } R_{ij} = & \eta_{n+i} p_j(\boldsymbol{x}) = \left(D_{\boldsymbol{u}_i} p_j\right)(\boldsymbol{y}_i).
\end{array}$$

When there is no polynomial part $k = 0$ and the block system is reduced by deleting the rows and columns containing Q and R. Note that $G = \left[\begin{smallmatrix} A & B^T \\ B & C \end{smallmatrix}\right]$ and $P = \left[\begin{smallmatrix} Q \\ R \end{smallmatrix}\right]$.

At first glance it may not be obvious that the top right section of G should be the transpose of the bottom left section, as the chain rule acts on the minus sign differently:

$$\begin{array}{ll}
\text{for } 1 \le i \le n,\ n+1 \le j \le m, & G_{ij} = -\left(D_{\boldsymbol{u}_{j-n}} \Phi\right)(\boldsymbol{x}_i - \boldsymbol{y}_{j-n}) \\
\text{for } n+1 \le i \le m,\ 1 \le j \le n, & G_{ij} = \left(D_{\boldsymbol{u}_{i-n}} \Phi\right)(\boldsymbol{y}_{i-n} - \boldsymbol{x}_j).
\end{array}$$

But since Φ is radial, we have

$$\Phi(\boldsymbol{x}) = \Phi(-\boldsymbol{x})$$

and therefore

$$\nabla\Phi(\boldsymbol{x}) = -\nabla\Phi(-\boldsymbol{x}),$$

implying

$$\left(D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) = -\left(D_{\boldsymbol{u}}\Phi\right)(-\boldsymbol{x}).$$

So for $n+1 \leq i \leq m$, $1 \leq j \leq n$ we have, in the top right section,

$$\begin{aligned} G_{ji} &= -\left(D_{\boldsymbol{u}_{i-n}}\Phi\right)(\boldsymbol{x}_j - \boldsymbol{y}_{i-n}) \\ &= \left(D_{\boldsymbol{u}_{i-n}}\Phi\right)(\boldsymbol{y}_{i-n} - \boldsymbol{x}_j) \\ &= G_{ij}, \end{aligned}$$

in the bottom left section. See Figure 7.2 for a visual demonstration of the evaluation types in G.

The Hermite basic functions are much easier to calculate than the integral equivalents from Part I. We need directional derivatives of radial functions, and then directional derivatives of those.

For a basic function $\Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|)$, where $\boldsymbol{x} = [x_1, \ldots, x_d]^T \in \mathbb{R}^d$, we have by the chain rule

$$\nabla\Phi(\boldsymbol{x}) = \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$$

and hence

$$\left(D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) = \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|}. \tag{7.7}$$

Then

$$\begin{aligned} \nabla\left(D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) &= \nabla\left(\phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|}\right) \\ &= \left(\nabla\phi'(\|\boldsymbol{x}\|)\right)\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|} + \phi'(\|\boldsymbol{x}\|)\left(\frac{\nabla(\boldsymbol{x}\cdot\boldsymbol{u})}{\|\boldsymbol{x}\|} - \frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|^2}\nabla\|\boldsymbol{x}\|\right) \end{aligned}$$

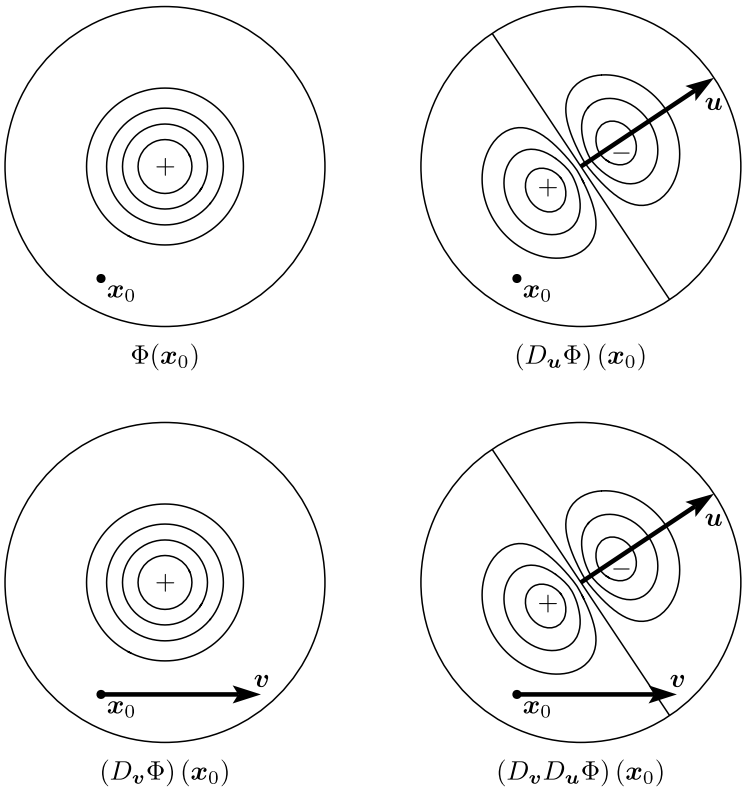


Figure 7.2: The two types of evaluation of the two types of basic function. Point evaluations at $\boldsymbol{x}_0$ (top) and directional derivative evaluations at $\boldsymbol{x}_0$ in direction $\boldsymbol{v}$ (bottom) of pointwise basic function $\Phi(\boldsymbol{x})$ (contours on left) and derivative basic function $(D_{\boldsymbol{u}}\Phi)(\boldsymbol{x})$ (contours on right). The contours are for the Wendland function $\Phi_{3,1}$.

(using the product rule)

$$\begin{aligned}
&= \phi''(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|} + \phi'(\|\boldsymbol{x}\|)\left(\frac{\boldsymbol{u}}{\|\boldsymbol{x}\|} - \frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|^2}\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\right)\\
&= \phi''(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|}\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} - \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\|\boldsymbol{x}\|^2}\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} + \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{u}}{\|\boldsymbol{x}\|}\\
&= \left(\frac{\phi''(\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|} - \frac{\phi'(\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|^2}\right)(\boldsymbol{x}\cdot\boldsymbol{u})\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} + \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{u}}{\|\boldsymbol{x}\|},
\end{aligned}$$

and therefore

$$\left(D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) = \left(\frac{\phi''(\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|} - \frac{\phi'(\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|^2}\right)\frac{(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})}{\|\boldsymbol{x}\|} + \phi'(\|\boldsymbol{x}\|)\frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\|\boldsymbol{x}\|}. \quad (7.8)$$

As this is symmetric with respect to $\boldsymbol{u}$ and $\boldsymbol{v}$, $D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi = D_{\boldsymbol{u}}D_{\boldsymbol{v}}\Phi$, as expected.

We present now Hermite forms derived from common radial basic functions by means of (7.7) and (7.8). Note that $\boldsymbol{x} = \boldsymbol{0}$ is best treated as a special case.

The work of Iske [51] contains sufficient conditions for a function to be strictly conditionally positive definite in a sense more general than that of our Definition 4. In particular, using his conditions on the Fourier and generalised Fourier transforms gives the HSPD orders shown here.

Gaussian

The Gaussian basic function is $\mathsf{HSPD}_{0,\infty}(\mathbb{R}^d)$ for all d.

$$\begin{aligned}
\Phi(\boldsymbol{x}) &= e^{-\nu^2\|\boldsymbol{x}\|^2}\\
\left(D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) &= -2\nu^2 e^{-\nu^2\|\boldsymbol{x}\|^2}(\boldsymbol{x}\cdot\boldsymbol{u})\\
\left(D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) &= -2\nu^2 e^{-\nu^2\|\boldsymbol{x}\|^2}\left(\boldsymbol{u}\cdot\boldsymbol{v} - 2\nu^2(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})\right)
\end{aligned}$$

Multiquadric

The negative of the multiquadric basic function is $\mathsf{HSPD}_{1,1}(\mathbb{R}^d)$ for all d.

$$\begin{aligned}
\Phi(\boldsymbol{x}) &= \sqrt{\|\boldsymbol{x}\|^2 + \nu^2}\\
\left(D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) &= \frac{\boldsymbol{x}\cdot\boldsymbol{u}}{\sqrt{\|\boldsymbol{x}\|^2 + \nu^2}}\\
\left(D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi\right)(\boldsymbol{x}) &= \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\sqrt{\|\boldsymbol{x}\|^2 + \nu^2}} - \frac{(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})}{(\|\boldsymbol{x}\|^2 + \nu^2)^{3/2}}
\end{aligned}$$

Thin-plate spline

The thin-plate spline basic function is $\mathsf{HSPD}_{2,1}(\mathbb{R}^d)$ for all d.

$$\begin{aligned}\Phi(\boldsymbol{x}) &= \|\boldsymbol{x}\|^2 \ln(\|\boldsymbol{x}\|) \\ (D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= (2\ln(\|\boldsymbol{x}\|) + 1)(\boldsymbol{x}\cdot\boldsymbol{u}) \\ (D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= (2\ln(\|\boldsymbol{x}\|) + 1)(\boldsymbol{u}\cdot\boldsymbol{v}) + \frac{2(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})}{\|\boldsymbol{x}\|^2}\end{aligned}$$

Cubic

The cubic basic function is $\mathsf{HSPD}_{2,1}(\mathbb{R}^d)$ for all d.

$$\begin{aligned}\Phi(\boldsymbol{x}) &= \|\boldsymbol{x}\|^3 \\ (D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= 3\|\boldsymbol{x}\|(\boldsymbol{x}\cdot\boldsymbol{u}) \\ (D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= 3\|\boldsymbol{x}\|(\boldsymbol{u}\cdot\boldsymbol{v}) + \frac{(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})}{\|\boldsymbol{x}\|}\end{aligned}$$

Wendland $\Phi_{3,1}$

The Wendland $\Phi_{3,1}$ basic function is $\mathsf{HSPD}_{0,1}(\mathbb{R}^3)$.

$$\begin{aligned}\Phi(\boldsymbol{x}) &= (1-\|\boldsymbol{x}\|/s)_+^4(4\|\boldsymbol{x}\|/s+1) \\ (D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= -20(1-\|\boldsymbol{x}\|/s)_+^3(\boldsymbol{x}\cdot\boldsymbol{u})/s^2 \\ (D_{\boldsymbol{v}}D_{\boldsymbol{u}}\Phi)(\boldsymbol{x}) &= -20(1-\|\boldsymbol{x}\|/s)_+^2\left((1-\|\boldsymbol{x}\|/s)\frac{\boldsymbol{u}\cdot\boldsymbol{v}}{s^2} + \frac{3(\boldsymbol{x}\cdot\boldsymbol{u})(\boldsymbol{x}\cdot\boldsymbol{v})}{\|\boldsymbol{x}\|s^3}\right)\end{aligned}$$

We now give the straightforward greedy algorithm we use to find a global approximation for the base level, Algorithm 4, as mentioned in Section 7.1. Its purpose is to produce a reasonable approximation to the dataset's basic shape, using on the order of a few hundred centres. The algorithm works iteratively, adding at each step the point from a candidate set with the worst residual (from 0, as all the points are on the surface). The candidate sets are cycled through; each contains points spread over the dataset, but few enough that the process proceeds quickly.

7.5 Sphere Coverings

We need to construct a set $\mathcal{S}$ of scattered overlapping spherical subdomains such that every point is in at least one sphere. Since we intend to build an RBF approximation in each sphere that takes all its internal points into account, a reasonable way to keep the difficulty of the subproblems similar is to pick a

Algorithm 4 Greedy RBF Fit for Base Level

Input : List of points $\mathcal{X}$ with corresponding normals $\mathcal{N}$, integer number of points to use n, basic function Φ, integer number candidate points to test in each iteration m

Output: Hermite RBF approximation σ_0 to $\mathcal{X}$ and $\mathcal{N}$ using n centres

1: Sort $\mathcal{X}$ and $\mathcal{N}$ into octree order. If necessary, reduce their size (to e.g. $200n$) by a simple thinning method such as only keeping every kth point for an appropriate k.
2: Choose a few (e.g. 10) initial points scattered through the dataset, and fit a Hermite RBF σ_0 to them and their normals.
3: Initialise test candidate offset j to 0.
4: **while** σ_0 uses less than n points **do**
5: Evaluate σ_0 at the points $\{\boldsymbol{x}_i \in \mathcal{X}$ such that $i = j \bmod \lfloor \#\mathcal{X}/m \rfloor\}$.
6: Update σ_0, adding the $\boldsymbol{x}_i$ whose residual has greatest magnitude (if > 0) along with its normal.
7: Increment j.
8: **end while**

number n, and ensure that every sphere contains n points. While doing this precise task for this precise goal is new, similar problems have been solved before.

Ohtake, Belyaev and Seidel have constructed scattered sphere coverings for two different purposes. In their papers of 2004 [86] and 2006 [89], the spheres are used as subdomains for a single level surface reconstruction method. Each sphere contains a quadratic approximation to the points inside it, and a compactly supported RBF is added where each basis function has one of the spheres as its support region. The sphere centres are chosen from the point set iteratively, requiring that the sum of the (Wendland) weight functions of already-chosen spheres, evaluated at the new centre, be less than a threshold (they suggest 1.5). A small set of points matching this requirement is chosen at random in each iteration, and the one with the lowest weight sum is selected as the centre of a new sphere. The radius of the new sphere is chosen by an optimisation procedure, depending on the local function approximation and confidence values assigned to the points.

This approach guarantees that every point will be covered by at least two spheres, but does not guarantee that a point will be near to the centre of any sphere, where the approximation is likely to be more accurate.

Also in 2006, Ohtake, Belyaev and Seidel [88] used a sphere covering for a completely different approach to surface reconstruction, based not on implicit function representation but on generating triangles directly. Inside each sphere a representative point is found (not usually a member of the point set), and the representative points of neighbouring spheres are connected to make triangles. The mesh is then automatically tidied up.

Again the covering proceeds iteratively, with a new centre chosen randomly

from the points not yet covered by any sphere. The radius is determined via an error optimisation procedure, tailored to the problem at hand and based on distances to tangent planes. Their covering also has convexity requirements which are unnecessary for our purposes. This method is related to a way of generating circular splats from a point cloud due to Wu and Kobbelt [131].

In a different area, Dunbar and Humphreys [24] cover the plane in disks as part of their two fast algorithms for Poisson-disk point sampling. That is, generating a set of points in the plane which are both scattered and roughly equidistant. Bridson [12] subsequently extended their approach to any dimension in a way which, while simple, may reduce the quality of the results as it depends on a statistical heuristic. These methods assume an empty domain to which points can be added arbitrarily, and involve constructing a disk (sphere) around each point to restrict and help to determine the locations of new points. In our case the spheres are the end goal and the points must be drawn from a particular fixed set. These methods also attempt to separate points by a fixed Euclidean distance, though Dunbar and Humphreys suggest extending this to a density based metric. In our case the metric is effectively density based, as it is simply the count of points inside a particular radius.

We need good overlap so that local approximations in different subdomains are similar where they will be blended together. At the same time, too much overlap will result in more subdomains than are actually needed, and hence more computational work.

Our straightforward sphere covering method, detailed in Algorithm 5 and demonstrated in Figure 7.3, has two key features which lead to even, well-packed and generally hole-free coverings. Firstly, we use a sweeping plane technique to position sphere centres near already generated spheres, and secondly, we use an active-core technique to improve the packing of the spheres. In the latter, points in a sphere are only marked as covered if they are in a spherical core with its radius smaller by a factor c.

Algorithm 5 Sphere Covering

Input : Set of points $\mathcal{X}$, integer points per sphere n, core proportion c
Output: Set of spheres $\mathcal{S}$ which cover $\mathcal{X}$

1: Create a list of points X by sorting the elements of $\mathcal{X}$ along one axis.
2: Mark all points as uncovered.
3: Initialise $\mathcal{S}$ to $\{\}$.
4: **while** uncovered points remain **do**
5: Choose the next uncovered point $\boldsymbol{x}_i$ from X as a centre.
6: Expand a sphere S_i around $\boldsymbol{x}_i$ until it contains n points.
7: Mark the points inside the core of S_i as covered.
8: Add S_i to $\mathcal{S}$.
9: **end while**

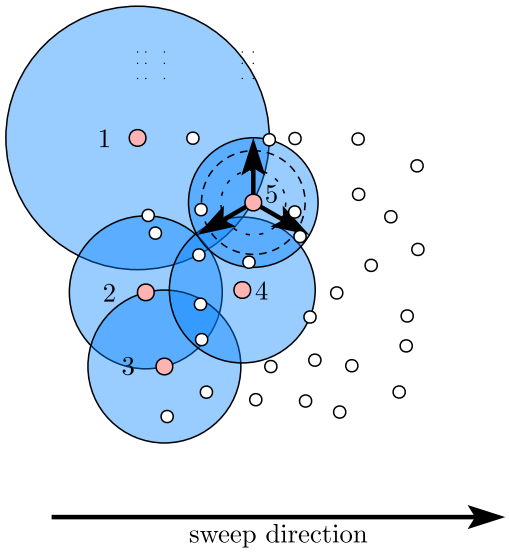


Figure 7.3: The sphere covering process. Here the covering is proceeding from left to right, with $n = 6$ points per sphere. (In practice we find $100 \leq n \leq 200$ works well, and we use such a small number here to make the diagrams clear.) For this figure, the core proportion c is set to 1.

Rather than choosing sphere centres at random from uncovered or insufficiently covered points like Ohtake, Belyaev and Seidel, we choose them to be close to the boundary of the partially completed covering. Since we are selecting points from a pre-defined set, we cannot choose a new point on the surface of a sphere, as in Dunbar and Humphreys' most efficient boundary sampling algorithm. However, since our points are in effect samples already taken, we can come close to this idea by simply choosing the next uncovered point from our set in the sorted order. With the sphere covering progressing across the domain in the direction of the sort axis, we are guaranteed that new sphere centres will be close to existing sphere boundaries, and that the newest spheres will be kept in lockstep near a plane sweeping through the domain.

The algorithms of [24] and [12] expand outwards in all directions from a start point, which works well for the problem of generating points, but is more complicated than our sweeping plane and difficult to adapt to the fixed points case. The sweeping plane approach also produces significantly better coverings than methods based on random point selection, with more consistent overlap and fewer holes.

Choosing one axis to sort along does add an arbitrary coordinate system dependence, but in practice this effect is not usually significant. One situation

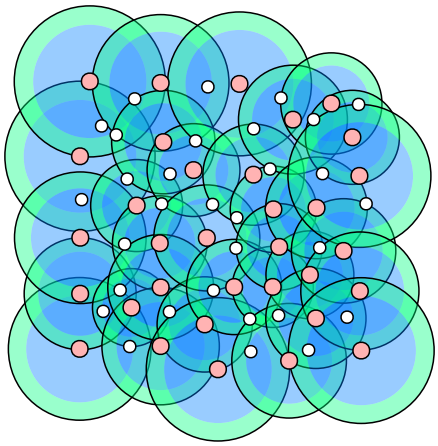


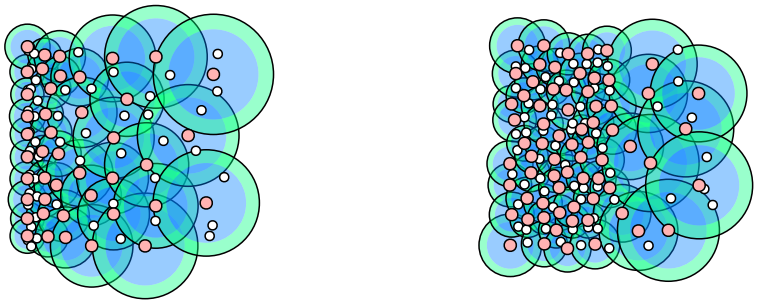
Figure 7.4: A complete sphere covering with 6 points per sphere and core proportion of $c = 0.75$.

where it could be a problem is if the dataset includes a large number of points at the same distance along the sort axis. When the sweeping plane reaches those points, the next few sphere centres will be chosen from them in whatever order they happen to be in, which could lead to a poorly packed covering in this region. This is easily avoided by choosing a different sort direction—it need not be a coordinate axis; sorting by the points' projections onto any line will work.

Ohtake, Belyaev and Seidel achieve full coverage by requiring that the sum of the subdomain weight functions exceeds a threshold at every point. While this method does not explicitly guarantee that points will be near the centre of at least one sphere, in practice it is very likely, as their recommended threshold is 1.5, and the maximum value of the weight functions is 1. The threshold condition also controls how tightly the spheres are packed, but with our sweeping plane technique ensuring tight packing, we are able to use a simpler method to achieve the coverage we need, with the idea of sphere cores.

We choose a constant proportion $0 < c \leq 1$ and define a sphere's *core* as a sphere having the same centre but with radius smaller by a factor of c (see Figure 7.4). When c is strictly less than one the requirement that each point is in the core of at least one sphere, rather than simply in some sphere, increases the amount of overlap in the covering and ensures that every point is in the more accurate central region of at least one subdomain.

Selecting sphere centres from the point set and having a fixed number of



(a) Gradual change (uniform density with the x component cubed).

(b) Abrupt change (uniform density with 80% of the points in the right hand half removed).

Figure 7.5: Sphere coverings of datasets with varying density, using 6 points per sphere and core proportions of $c = 0.75$.

points inside each sphere simplify the process by leaving only one easily found choice for the next centre. Further, they make the covering explicitly and wholly data driven rather than space or geometry driven. With these choices in place, the process of creating subdomains is completely decoupled from the process of constructing approximations within the subdomains, and this feature of our method turns out to be important for the ability to scale to very large datasets, as we explain in Section 7.8.

Our algorithm generates sphere coverings that are automatically density dependent, and easily copes with both sudden and gradual density changes (see Figure 7.5).

During the covering process, a set of points is associated with each sphere, and these are then used in the construction of the small local approximations. When we are evaluating the finished piecewise function the story is different, since the evaluation point is probably not in the original dataset. We therefore need a means of determining which spheres any given point is inside. This is a straightforward job in grid-based piecewise techniques such as MPU (Multi-level Partition of Unity) [83] or the method of Tobor, Reuter and Schlick [114, 115, 116], but since our subdomains are scattered we have a slightly trickier problem and use the following basic data structure, which we refer to as a sphere tree.

The domain of interest is placed inside a cube which is recursively divided into octants, forming an octree, dividing a node so long as there are more than n_s (a fixed parameter) spheres intersecting a subcube C, and C's octants intersect fewer spheres than C does. Each leaf node has a short list of spheres associated

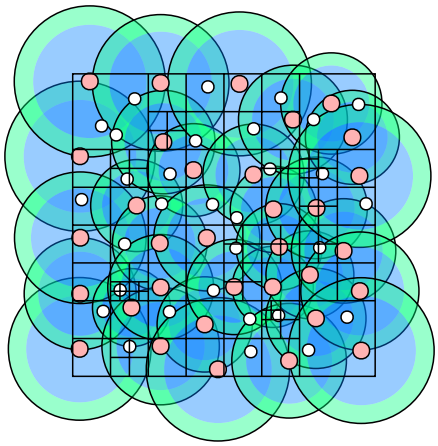


Figure 7.6: The sphere tree of the sphere covering from Figure 7.4, with $n_s = 5$.

with it, and spheres are stored only at leaf nodes. This structure is constructed with Algorithm 6, and illustrated in Figure 7.6.

To find out which spheres contain a point $\boldsymbol{x}_0$, we drill down into the sphere tree, choosing at each node the child node whose subcube contains $\boldsymbol{x}_0$. When we reach a leaf node, we run through its list of spheres, selecting those which contain $\boldsymbol{x}_0$. In practice we find a good value for n_s to be 300; the length of the lists to be searched needs to be balanced against the depth of the tree.

Algorithm 6 Sphere Tree Construction

Input : Set of spheres $\mathcal{S}$, cube C, integer spheres per leaf-node n_s
Output: Sphere tree $\mathcal{T}$

Create a tree node r and associate it with C.
if $\#\mathcal{S} > n_s$ **and** each octant of C intersects a strict subset of $\mathcal{S}$ **then**
  for each octant C_i of C **do**
    Call this algorithm with $(\{s \in \mathcal{S} \text{ s.t. } s \text{ intersects } C_i\}, C_i, n_s)$.
    Make the resulting subtree a child of r.
  end for
else
  Make r a leaf node and associate it with $\mathcal{S}$.
end if
return the tree $\mathcal{T}$ with root node r.

While for "nice" enough datasets holes are very seldom seen, they are not explicitly prevented, as demonstrated in Figure 7.7. This stems from the difficulty in even defining a "hole" for datasets of discrete points such as these.

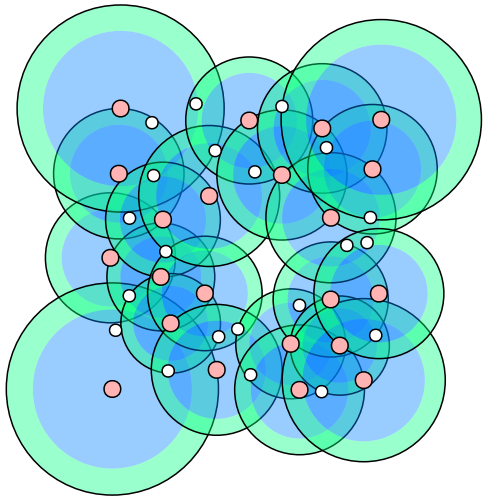


Figure 7.7: A sphere covering where an (artificial) unusually large space leads to a hole.

Holes like this are problematic because they represent regions of our domain of interest where the surface function will be undefined. In the multilevel situation, while we have a lower level to fall back on, a partition of unity will create a discontinuity at the hole's boundary. These issues often lead to holes or glitches in the reconstructed surface, or spurious extra bits of surface.

As observers with knowledge about the underlying shape a point set is created from, we find it easy to determine whether a hole is present in error, but to an algorithm with only the points themselves to go on, the problem is extremely difficult. Thus we cannot rely on the sphere covering to be hole-free, and so we embed this method within a system which can handle any holes which might be produced.

We have two such systems, one based on late hole detection (Section 7.6), and the other on modifying the partition of unity method to include level blending (Section 7.7).

7.6 Late Hole Detection

In this section we describe a method for detecting and recovering from holes in the sphere covering called Late Hole Detection (LHD). Detecting holes from points and their normals directly is difficult, but it becomes much easier once a surface approximation exists. If the calculated surface runs right up to the

edge of a sphere in a region not overlapped by another sphere, then this forms the edge of a hole.

Of course, searching for the surface like this is an expensive process involving many function evaluations. Much of the time, however, the desired result of the surface reconstruction is a polygonisation of the model, and the process of polygonisation is exactly the process of determining where the surface goes, to a certain resolution.

For polygonisation we use the Single-Entry Cubical Table method described by Ning and Bloomenthal in [82]. This is a surface following method which evaluates the surface-defining function on a fixed-size cubical grid. Starting at a seed point (or points) on the surface, it evaluates at the corners of a grid cell, produces triangles approximating the surface within that cell, then determines which of its neighbours the surface enters and continues recursively into them. A set of boundary grid cells is maintained, and at each iteration these are evaluated and the boundary set is advanced into neighbouring cells which the surface enters. In this way, the polygoniser will explore all connected parts of the seeded surface.

The exact surface following method is not important, but the distances between evaluations are (and these are usually connected to the sizes of the final triangles). With larger distances the polygoniser may step harmlessly over small holes. At the same time, however, larger distances may mean that the polygoniser may step out of the sphere covering completely simply because it is getting too far away from the points on the surface.

The basic idea of the LHD method is to detect the edges of holes and add extra spheres as necessary which blend down into the level below, and let that level take care of filling the hole (or pass it down further, possibly as far as the globally supported base level). The process is shown in Figures 7.8 and 7.9, summarised in Algorithm 7, and described in detail in the text now.

We need to detect holes both as we are stepping into them and as we are stepping out of them, and we do this by defining a "trigger margin" near the edges of the spheres as

$$\{\boldsymbol{x} \text{ such that } 0 < \sum_i w_i(\boldsymbol{x}) < \epsilon\},$$

where the w_i are the partition of unity weight functions and ϵ is a constant threshold. This is free to calculate, since we need the same sum of weight functions as part of the partition of unity evaluation. The trigger margin needs to be wider than the grid spacing, so that it cannot be stepped over, so ϵ does need to be chosen carefully at times. A useful future modification may be to choose ϵ dynamically or on a per-sphere basis, to maintain a minimum margin

width. We set the trigger conditions for adding a new sphere at evaluation point $\boldsymbol{x}$ to be:

1. $\boldsymbol{x}$ is inside the trigger margin, and
2. the partition of unity evaluation for the current level $\sigma_{\mathrm{L}}(\boldsymbol{x}) = 0$.

The second condition will be explained shortly.

When the polygoniser steps onto a point where these conditions are met, a trigger point, we say it has found a hole in the covering. Note that while the points are on a 2-dimensional surface, the covering needs to have a definite thickness, so that the polygoniser can probe numerically to find the surface. A "hole", therefore, need not be a gap all the way through the covering from one side of the surface to the other.

We recover from these situations by adding a new sphere, centred at the trigger point and expanded, as before, until it contains n points. In this new sphere we create a surface approximation, which, to accomplish the blending, must use a compactly supported basic function with a support radius small enough that the approximation will die away to zero by the far edge of the sphere, and have no polynomial part. Since the new sphere is at the edge of a hole, its data points are usually located only in one hemisphere, and so choosing the approximation's support radius to be half the sphere's radius provides a good compromise between dying away and approximating well. The second trigger condition above takes advantage of the compact support to prevent us from continuing to add extra spheres building on each other out into the hole.

Adding an extra sphere extends the piecewise function's domain, but that is not quite enough to get things back on track. Since the complete surface approximation function consists of a partition of unity combination of the local approximations defined in the spheres, adding a new sphere will change it, and thus invalidate every previous evaluation that occurred in the region now covered by the new sphere.

This has two important consequences for the polygonisation process. Firstly, it must be possible to invalidate triangles or to avoid generating possibly invalid triangles, and secondly, there must be a way to redo the surface following in invalidated regions.

We choose to implement the polygoniser as a two stage process, first following the surface on the evaluation grid, dealing with holes and extra spheres as they are encountered, and then producing triangles inside the already evaluated grid subcubes. With this approach, grid points may be redone but not triangles, since the polygoniser does not produce triangles until all the evaluations have been completed and there is no chance that the evaluations they depend on will be invalidated.

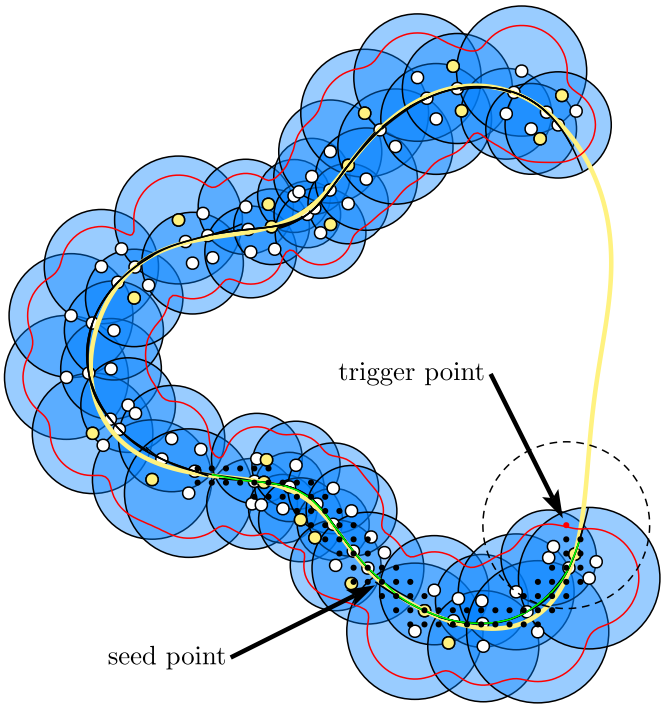


Figure 7.8: The Late Hole Detection process. The polygoniser has advanced in both directions from the indicated seed point, producing the green surface reconstruction by evaluating the multilevel partition of unity approximation at the black grid points. In this figure, the polygoniser has just stepped over the red trigger margin onto a red trigger point, and consequently a new sphere, dashed, is being added. The approximations used here are, for demonstration purposes, pointwise rather than Hermite, and the points are shown as white or yellow dots. The yellow global approximation line is constructed from the yellow points. Inside each sphere is a local approximation shown as a black line, though these are often on top of each other or underneath the green surface line. Note that the black local approximations deviate from the yellow global approximation, which they are refining, and also from each other, where spheres overlap. This figure distorts scales for comprehensibility, with only 7 points per sphere, an over-large grid spacing, and a large proportion of points used for the base level.

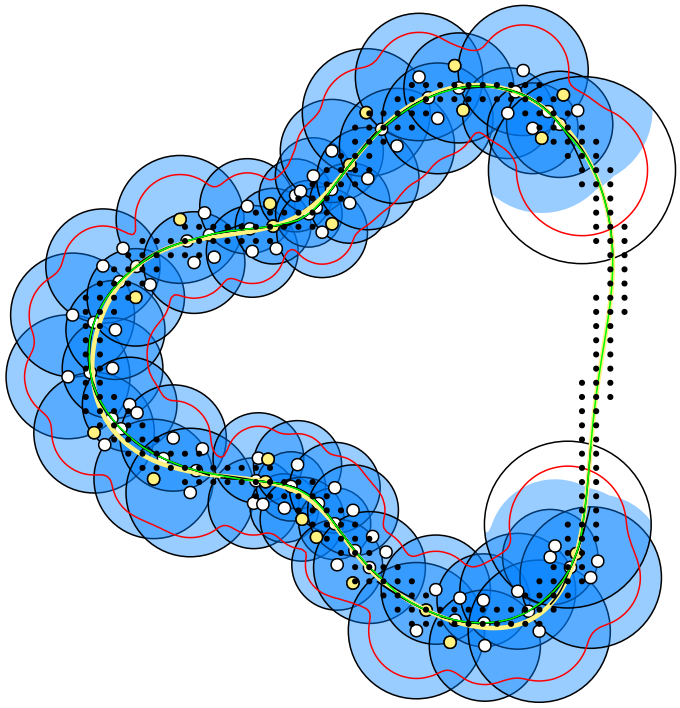


Figure 7.9: The complete surface reconstruction by Late Hole Detection. Note that the support of the local approximations, blue, extends only about halfway across the two extra spheres, and that outside the blue regions the green surface reconstruction simply follows the yellow global approximation.

During the surface following stage, the polygoniser stores every grid location it has visited along with their function values in a sparse octree data structure. While this can use a lot of memory, it simplifies the implementation and there are good reasons to do it this way. To avoid duplicating parts of the surface, the polygoniser needs to remember at least its recent history, but keeping track of which points to remember would rely on a potentially fallible heuristic such "the last N points visited" or "points visited in the last M iterations". Leaving open the possibility of invalidating and redoing past evaluations also means storing points. It can be argued that recent points are enough here as well, since if a very old point falls within the sphere being invalidated, it is probably not closely connected to the region of the newly discovered hole, and thus its existing evaluations still make sense without taking the new sphere into account. Nevertheless, it is not clear how to choose the amount of history to remember, and so we stay with the simplest approach of remembering everything. For large datasets we apply a memory-saving strategy based on the piecewise nature of the approximation, which we explain in Section 7.8.

Once the grid points inside a new sphere have been invalidated, the surface following in these regions needs to be redone taking the newly modified surface function into account. The function remains the same on the edges of the new sphere, so we know that the modified surface still enters it, and we know that there are surface points inside the new sphere, by its construction. Thus to restart the surface following inside the new sphere we simply add to the boundary set all grid cells which contain data points inside the sphere. An alternative for the non-interpolating case is to add the grid cells which intersect the surface of the sphere and which, by their neighbours' corner values, the approximated surface enters.

Adding extra spheres is an inevitably expensive process, whose occurrence is minimised and kept to "real" holes by careful choice of sphere covering parameters.

Algorithm 7 Surface Reconstruction With Late Hole Detection

Input : Set of points $\mathcal{X}$, integer points per sphere n, grid size $\left(\frac{1}{2}\right)^m$
Output: Set of triangles $\mathcal{T}$ which approximate the surface underlying $\mathcal{X}$

1: Use Algorithm 5 to generate a set of spheres $\mathcal{S}$ covering $\mathcal{X}$ with n points per sphere.
2: Construct a local approximation inside each sphere, and combine these with a partition of unity to create an approximation $s(\boldsymbol{x})$.
3: Initialise an empty grid G.
4: Choose one or more seed points from $\mathcal{X}$ and initialise the boundary set $\mathcal{B}$ with the cells of G which contain them.
5: **while** $\mathcal{B}$ not empty **do**
6: **repeat**
7: **try**
8: Evaluate s at the corners of the cells in $\mathcal{B}$, storing the results in G and marking the cells as visited.
9: **catch exception** found trigger point $\boldsymbol{x}_i$
10: Add a new sphere $\mathcal{S}_i$ centred at $\boldsymbol{x}_i$ and large enough to contain n points from $\mathcal{X}$.
11: Construct a local approximation inside $\mathcal{S}_i$ using a compactly supported basic function with support radius half that of $\mathcal{S}_i$.
12: For each cell of G which intersects $\mathcal{S}_i$, remove any previous evaluation result and mark as unvisited.
13: Find the cells of G which contain the points of $\mathcal{X}$ inside $\mathcal{S}_i$, and add them to $\mathcal{B}$.
14: **end try**
15: **until** all corners of cells in $\mathcal{B}$ evaluated
16: Find the set of neighbours of cells in $\mathcal{B}$, $\mathcal{N}$.
17: Replace $\mathcal{B}$ with the members of $\mathcal{N}$ that are unvisited and that the surface enters.
18: **end while**
19: Construct $\mathcal{T}$ by running through the visited cells of G and generating triangles to fit the stored function values.

7.7 Level-Aware Partition of Unity

In this section we present a modification of the standard partition of unity method of Section 7.2 which produces a smooth blend down to lower levels at the edges of spheres. This modification is called Level-Aware Partition of Unity (LAPOU), and it works by adding the zero function into the partition of unity as an extra term, with a constant weight. With this change, (7.1) becomes

$$\sigma_{\mathrm{L}}(\boldsymbol{x}) = \frac{\omega_{\mathrm{L}}\theta(\boldsymbol{x}) + \sum_i w_i(\boldsymbol{x})s_i(\boldsymbol{x})}{\omega_{\mathrm{L}} + \sum_i w_i(\boldsymbol{x})}, \tag{7.9}$$

where ω_{L} is a positive constant and θ is the function that is everywhere zero. The expression therefore simplifies to

$$\sigma_{\mathrm{L}}(\boldsymbol{x}) = \frac{\sum_i w_i(\boldsymbol{x})s_i(\boldsymbol{x})}{\omega_{\mathrm{L}} + \sum_i w_i(\boldsymbol{x})} = \sum_i v_i(\boldsymbol{x})s_i(\boldsymbol{x}), \tag{7.10}$$

where we have redefined $v_i(\boldsymbol{x})$ to include the ω_{L} term. Note that now in general $\sum_i v_i(\boldsymbol{x}) \neq 1$.

For a sphere $\mathcal{S}_i$ with an edge region not covered by other spheres, as we move towards that edge there will be only one positive sphere weight function, $w_i(\boldsymbol{x})$, which will decrease to 0. In the usual partition of unity, the weight diminishes equally to 0 on the top and bottom of the expression, so the local approximation s_i is left unchanged. It then jumps abruptly to zero when we cross the boundary of the sphere covering, as the approximation is only defined inside it. In our modified version, the bottom weight decreases to the positive constant ω_{L}, so that σ_{L} goes smoothly to 0, automatically avoiding the discontinuity phenomenon.

Clearly, this will break interpolation if the usual approximation conditions are used, so they need to be modified correspondingly. The requirement $\sigma_{\mathrm{L}}(\boldsymbol{y}) = r_{\mathrm{L}}(\boldsymbol{y})$ at the data points of level L implies

$$\sum_i v_i(\boldsymbol{y})s_i(\boldsymbol{y}) = r_{\mathrm{L}}(\boldsymbol{y}). \tag{7.11}$$

When we require, as before, that the local approximations agree at the data points in overlapping regions, this time setting $s_i(\boldsymbol{y}) = \rho_{\mathrm{L}}(\boldsymbol{y})$ for each i such that $\boldsymbol{y} \in \mathcal{S}_i$, we obtain

$$\rho_{\mathrm{L}}(\boldsymbol{y}) \sum_i v_i(\boldsymbol{y}) = r_{\mathrm{L}}(\boldsymbol{y}), \tag{7.12}$$

implying

$$\rho_{\mathrm{L}}(\boldsymbol{y}) = \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})}. \tag{7.13}$$

So ρ_{L} is a modified version of the residual which undoes the weighted scaling of the local approximations.

Similarly to the conventional partition of unity case, if at a data point $\boldsymbol{y}$ we know the directional derivative $(D_{\boldsymbol{u}} r_{\mathrm{L}})(\boldsymbol{y})$, then for each i such that $\boldsymbol{y} \in \mathcal{S}_i$ we set both $s_i(\boldsymbol{y}) = \rho_{\mathrm{L}}(\boldsymbol{y})$ and $(D_{\boldsymbol{u}} s_i)(\boldsymbol{y}) = (D_{\boldsymbol{u}} \rho_{\mathrm{L}})(\boldsymbol{y})$, where

$$(D_{\boldsymbol{u}} \rho_{\mathrm{L}})(\boldsymbol{y}) = \frac{(D_{\boldsymbol{u}} r_{\mathrm{L}})(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})} - \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\left(\sum_i v_i(\boldsymbol{y})\right)^2} \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y})$$

is the directional derivative of (7.13), using the product rule. From the directional derivative of (7.10) we obtain

$$\begin{aligned}
(D_{\boldsymbol{u}} \sigma)(\boldsymbol{y}) &= \sum_i v_i(\boldsymbol{y}) (D_{\boldsymbol{u}} s_i)(\boldsymbol{y}) + \sum_i s_i(\boldsymbol{y}) (D_{\boldsymbol{u}} v_i)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} \rho_{\mathrm{L}})(\boldsymbol{y}) \sum_i v_i(\boldsymbol{y}) + \rho_{\mathrm{L}}(\boldsymbol{y}) \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) \\
&= \left(\frac{(D_{\boldsymbol{u}} r_{\mathrm{L}})(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})} - \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\left(\sum_i v_i(\boldsymbol{y})\right)^2} \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) \right) \sum_i v_i(\boldsymbol{y}) \\
&\quad + \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})} \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} r_{\mathrm{L}})(\boldsymbol{y}) - \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})} \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) + \frac{r_{\mathrm{L}}(\boldsymbol{y})}{\sum_i v_i(\boldsymbol{y})} \left(D_{\boldsymbol{u}} \sum_i v_i \right)(\boldsymbol{y}) \\
&= (D_{\boldsymbol{u}} r_{\mathrm{L}})(\boldsymbol{y}),
\end{aligned}$$

as expected.

In the definition of ρ_{L} in (7.13), the denominator of the fraction, $\sum_j v_j(\boldsymbol{y})$, goes to 0 at the boundary of $\cup \mathcal{S}_i$. Therefore, in order for these new interpolation conditions to remain sensibly achievable, no data point can be allowed too close to the boundary. To ensure this we set the constant c of section 7.5, defining the core proportion, to be strictly less than one. In practice we find good values to be $35\% \leq c \leq 50\%$, and our experience is discussed in Section 8.2.

The modified residual at a data point $\boldsymbol{y}$ depends on the weight functions of all the spheres which contain $\boldsymbol{y}$, and so at least the local part of the sphere covering needs to be complete before local approximations covering $\boldsymbol{y}$ can be constructed. New spheres cannot be added to regions where local approxima-

tions exist without invalidating them. So as before, the sphere covering and approximation phases are separate, but in this approach each local approximation depends explicitly on all the spheres which intersect its sphere, unlike in the late hole detection method of Section 7.6, where the approximations are independent of other spheres.

The LAPOU method works for a wide range of ω_L values, with larger values increasing the influence of the level below and providing smoother blending at the edges of holes. This greater influence can be problematic when the polygonisation grid size is large relative to the top level spheres, in which case we use a smaller ω_L. $0.005 \leq \omega_L \leq 5$ is a reasonable range, and our default is 1, though we find 0.05 to be good for large hole-less datasets. See Figure 8.10, page 109.

Level-Aware Partition of Unity is generally better than Late Hole Detection. If one's data can be guaranteed to be free of holes, then the standard partition of unity used by LHD would be slightly faster and thus preferable, but then of course there would be no need for LHD's extra spheres. In practice such datasets seem to be very uncommon, since any crevice between spheres can count as a hole if the polygoniser happens to step into it. In our opinion, the cost incurred when an LHD extra sphere is added and the considerable additional implementation complexity required to support it outweigh the rather small cost of LAPOU's residual adjustment and its slightly higher tendency to produce blobby artefacts when polygonisation grid and sphere sizes are close (see Section 8.2).

7.8 Large Datasets

Our primary motivation for developing piecewise methods is to be able to handle large datasets, and the smaller size of the approximation problems goes a long way towards that end. While a reconstruction using our methods can involve many thousands of spheres, the sphere covering process is fast and the local approximations are fast to fit and evaluate. Still, for larger datasets memory requirements become an important consideration, and we have two complementary strategies which directly address this issue. The first, On-the-fly Fitting, reduces the number of local approximations held in memory at any time, and the second, Block-Based Polygonisation, reduces the memory requirements of the polygoniser. We also describe two possible future directions, parallelisation and a lockstep plane technique.

7.8.1 On-the-fly Fitting

The sphere covering is fast to do, and takes little space to store, since all that is required are the sphere's centre and radius, and a sphere tree structure with which to access them quickly. The local approximations inside the spheres do require a fair amount of memory, however, as each one needs a set of points for RBF centres and a set of coefficients for the corresponding basis functions. Since multiple spheres share points and directions where they overlap, we can save memory by storing the coordinates of all the points and directions in single array (which itself may be hundreds of megabytes), and just storing point indices for the RBF centres.

We can do better than this though, by taking into account the way the the approximations are used. The polygoniser begins at one or more seed points and maintains a boundary set which is advanced across the surface, passing through spheres and leaving behind evaluated grid points. Because the approximations are local, they are only needed while the polygoniser is advancing through their spheres. This will almost always happen only once, as spheres containing multiple separate pieces of surface are rare.

In on-the-fly fitting, we maintain a pool of the N most recently used local approximations, fitting them as needed and forgetting the oldest as new ones come in. Most approximations which represent multiple pieces of surface will be remembered long enough for all their separate parts to be visited, as the pieces are usually close to each other on the surface (opposite sides of a fold, for example). Very occasionally a sphere may be visited again after its approximation has been forgotten, but this presents no problem, since fitting an individual local approximation is very cheap, and the approximations are independent and completely determined by the geometry of the sphere covering; the re-fit will be identical to the original.

7.8.2 Block-Based Polygonisation

The sparse octree system is a memory-efficient way of storing visited grid points and their function evaluations, as is the hash table approach used by some researchers (e.g. Ohtake et al with MPU [83]), but for small polygonisation grid sizes the sheer number of evaluation points makes remembering them all prohibitive. (Large datasets mean smaller spheres which means a smaller grid spacing.) As mentioned in Section 7.6, there are alternatives to storing every evaluation point, and here we describe a block-based technique which takes advantage of the piecewise nature of the approximation, and also happens to fit in nicely with one possible approach to parallelisation, described in the next section.

Polygonisation is usually conducted within a bounded volume, to prevent non-manifold surfaces or wayward hole filling from running away indefinitely. In block-based polygonisation we simply split this volume up into multiple smaller blocks and perform the same polygonisation as before within each of them, treating them completely independently. For blocks we use axis-aligned cuboids whose corners are points from the polygonisation grid.

Spheres which intersect more than one block will be shared between them, and their approximations will be reproduced in the separate block processes. As long as there are relatively few blocks, these shared spheres will make up a small proportion of the total. Because the local approximations depend only on the sphere covering, the approximations in the shared spheres will be identical when recalculated in separate blocks, thus making the blocks' surface reconstructions match up perfectly at the join.

It is important that each block contain only one contiguous piece of surface, or that each block be given sufficient seed points to reach every part of the surface within it. For this reason, blocks in our current implementation are manually specified, and for most models this is quick and easy to do. If automatic block generation is to succeed, automatic seeding is required, and unsophisticated methods, such as seeding at every grid cell which contains a data point, will work only if no hole filling from a neighbouring block protrudes into this one. Further, that kind of brute force approach will likely negate any benefit from using on-the-fly fitting. With a good seeding method it may be possible to reproduce those benefits by simply having a small total number of spheres per block—of course, then other considerations like the number of shared spheres will likely become important.

7.8.3 Parallelisation

Our current implementation is completely serial, but the methods are definitely parallelisable and parallelisation is an important future direction. There are several different ways to parallelise aspects of sphere-based piecewise surface reconstruction, acting at different stages in the process and various levels of the system.

Generating a sphere covering is one of the quickest stages in the process, and our method as described is not parallelisable directly. Nevertheless, it may be possible to create a parallel version using ideas similar to those of Wei's parallel Poisson disk sampling method [124], where regions too far apart to influence each other are sampled simultaneously. As the timing results in Chapter 8 will show, this is not a high priority.

Once the sphere covering exists, points and spheres are accessed via point

and sphere trees, which are read-only (except for when a sphere is added in LHD, another reason for LAPOU to be preferable). When on-the-fly fitting is not being used, the process of fitting the local approximations is trivially parallelisable, as they are completely independent from each other. When on-the-fly fitting is being used, the parallelism of the fitting is (mostly) driven by the parallelism of the evaluation, and the pool of approximations needs to be made thread-safe. While in principle new spheres which are entered simultaneously can have their approximations fitted simultaneously, the record keeping machinery that keeps track of which spheres have approximations and which were visited most recently needs to be kept single-threaded, or accessible from only one thread at a time.

At a lower level, fitting a local approximation involves evaluating the multi-level approximation below the current at the data points, to find the residuals. These evaluations are independent and could thus be done in parallel. At a lower level still, each RBF evaluation involves evaluating a large number of basis functions, and *this* could be done in parallel.

Working our way back up, at each iteration of the polygonisation process, the previously unseen corners of the grid cells in the boundary set need to be evaluated, another parallelisable task. When the polygoniser has finished following the surface the final step is to generate triangles from the stored function values. Since the triangles inside a subcube depend only on the cube's corner values, the subcubes are independent and could be processed in parallel.

The polygonisation process as a whole could be parallelised by means of the blocks described above. Since each block is self-contained and the shared spheres ensure that the edges match up, the blocks can be processed sequentially, simultaneously on different cores, or even on separate computers. No shared memory is strictly necessary, as the sphere covering is cheap enough to be recalculated if necessary, though when running on a single computer it would be advantageous to share the point and sphere data structures. This is a much higher level and more versatile form of parallelisation than the other possibilities mentioned above.

7.8.4 Lockstep Plane

The sweeping plane approach of our sphere covering algorithm (see Section 7.5, 75) could be taken for the fitting and polygonisation stages as well. All three processes could then proceed in lockstep with each other, sweeping across the domain and minimising memory usage.

Making the fitting process sweep with the covering is trivial—simply generate local approximations as soon as their spheres are created, and forget them

once they have been passed—but the polygonisation is another story. The basic idea is to step the plane forward one planar slice of grid cells at a time, fully polygonising each slice and generating triangles before moving on. This means surface following within a slice before generating an initial boundary set for the next slice. When the next slice is entered, spheres and approximations are generated if the slice contains uncovered points, and the process repeats.

We have a prototype implementation of these ideas which shows promise, but there remains one tricky issue. What happens when part of the surface protrudes back into the current slice without being attached to anything yet seen? This is similar to the automatic seeding problem discussed under block-based polygonisation above. Much of the time it will be solved by adding the grid cells of newly met data points to the boundary set, or perhaps by adding them a slice ahead, in anticipation. Sometimes, however, the surface will protrude back without data points (or even spheres) being present, and this will necessitate some form of backtracking. Parallelism could be added to the lockstep plane approach in any of the lower level ways mentioned above, or through blocks, especially if the covering was done first.

Chapter 8

Applications and Results

In this chapter we present a variety of results about the behaviour of the sphere-based piecewise surface reconstruction method.

Our implementation is written primarily in C++ [109], in the form of modules to be controlled from Python [123] programs. We use SciPy's [55] arrays and array-based operations, and solve our linear systems with the Intel Math Kernel Library's [50] implementation of LAPACK [3], running single-threaded. The compiler we use is the Intel C++ Compiler icpc [49].

8.1 Covering Behaviour

In this section we analyse the sphere coverings produced by our algorithm using a single model, the Stanford Bunny. As our interest here is in producing comparable and easily understood examples we use a small and well-behaved dataset, the points from the relatively low resolution reconstruction `bun_zipper.ply`, produced by the Stanford University Computer Graphics Laboratory[1]. Of the 35,947 vertices in that file, we use the 34,834 that are part of at least one face. We scale the points to fit inside the unit cube with a margin of 0.05 on all sides, centre them, and estimate a normal for each point as the average of the normals of the faces incident to it. These points are shown in Figure 8.1.

For the most part the points are very evenly spaced, which will lead to coverings with similarly sized spheres. Of note are the holes on the bottom, two long thin ones on the left and near edges of the inset section, and two large round ones on either side of it. Not visible is a small hole between the front legs.

[1] From the package `bunny.tar.gz`, available from the Stanford 3D Scanning Repository: `http://graphics.stanford.edu/data/3Dscanrep/`.

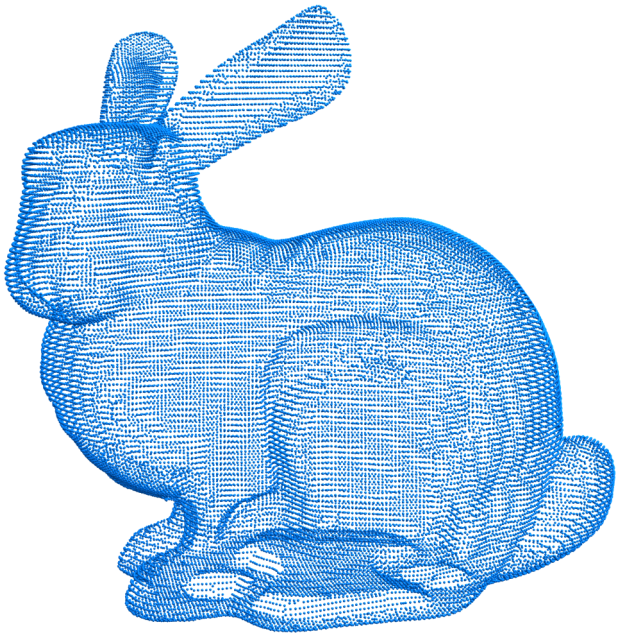


Figure 8.1: The points of the Stanford Bunny test dataset.

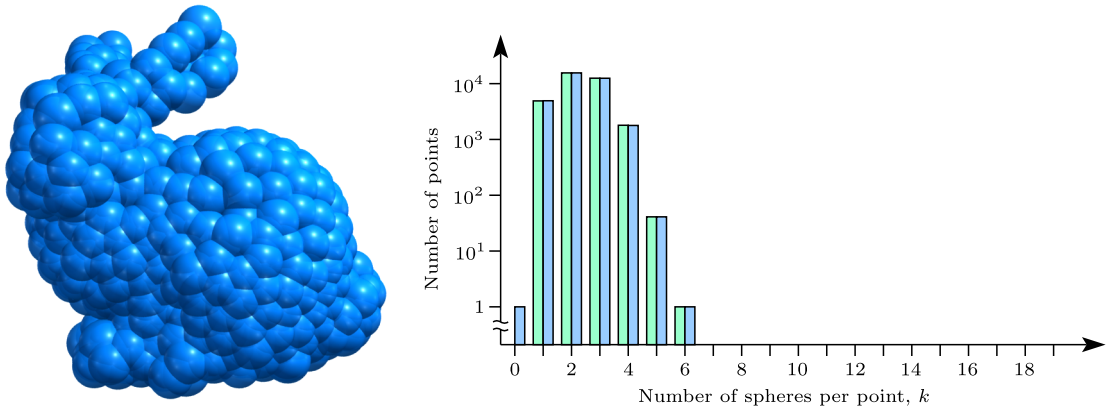
From Section 7.5, our sphere covering method has two parameters, the number of points per sphere n, and the core proportion c. n controls how big (hence expensive) the approximations inside the spheres will be, and c controls how much the spheres overlap. Both parameters affect the total number of spheres which will be produced.

Several coverings with n fixed at 200 and varying c are shown in Figure 8.2. As the amount of overlap increases, the number of spheres increases dramatically, but the denser coverings reduce and all but eliminate the deep crevices between spheres visible for $c = 1$. As a measure of overlap, we count the number of spheres covering each point, and show these values in the bar charts on the right. For example, the first non-zero pair of columns in (b) represent the number of points covered by only 1 sphere (64) and the number of points inside the core of only 1 sphere (5,516).

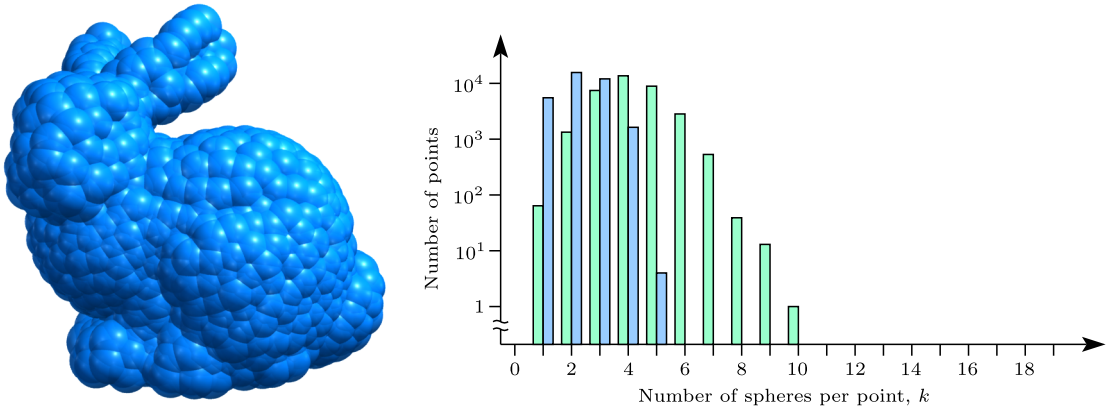
Note that the pattern of the bars for the points covered by cores (blue) remains roughly the same while the pattern for the full spheres (green) spreads out and shifts to right. By $c = 0.5$, only one point covered by fewer than 5 spheres remains, and some points are inside as many as 19 spheres. Within each covering is a "shadow" covering consisting of just the cores. While the cores contain variable numbers of points, unlike the spheres, the number of points in each core will be similar, as the core sizes are tied directly to the sphere sizes. Hence the shadow coverings behave like a covering with $c = 0$ (and an n depending on c), and the bars for the cores form a similar pattern for different values of c.

This pattern breaks down as the absolute core sizes near the distances between points, as in the coverings of Figure 8.3. The bar chart for Figure 8.3b is much too stretched out to show here, with no point in fewer than 125 spheres, and several in well over 300, but with 34,775 spheres for 34,834 points, the cores are so small that only 4 points are inside two of them. At this point we have very nearly moved back to a modified Shepard's method.

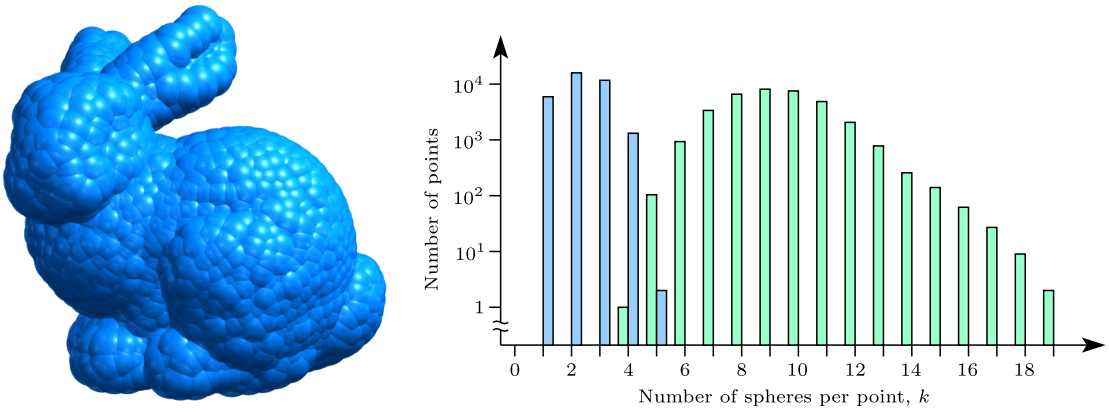
We have observed experimentally for a wide range of datasets that the number of spheres produced by our algorithm for $c = 1$, as a proportion of the total number of points, lies very close to $2/(1 + n)$. We do not yet have a theoretical justification for this relationship, but its inverse can be used as a heuristic which, together with an auxiliary sphere covering, makes a simple even thinning method. In practice we use either this or the octree order method mentioned in Algorithm 4, page 76, to obtain the subsets of points used for different levels.



(a) Core proportion $c = 1$: 405 spheres.

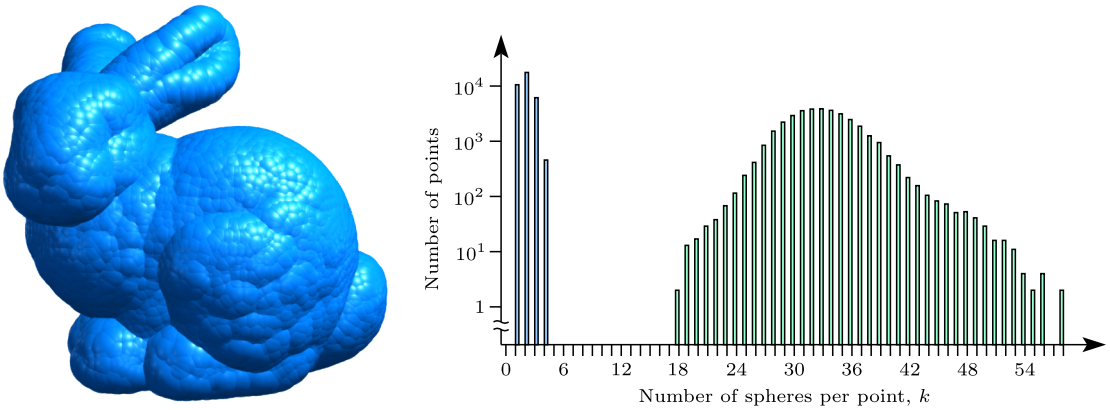


(b) Core proportion $c = 0.75$: 727 spheres.

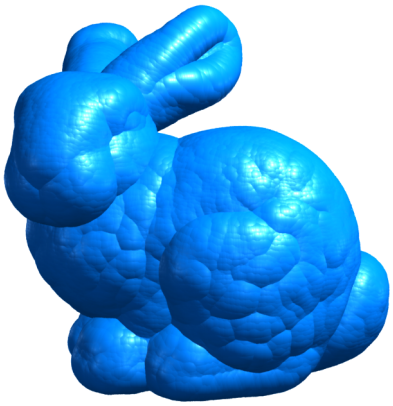


(c) Core proportion $c = 0.5$: 1,632 spheres.

Figure 8.2: Sphere coverings of the Stanford Bunny, with $n = 200$ points per sphere. The bar charts show the numbers of points covered by the whole, green, and the core, blue, of exactly k spheres, for several values of k.



(a) Core proportion $c = 0.25$: 5,772 spheres.



(b) Core proportion $c = 0.05$: 34,775 spheres.

Figure 8.3: Sphere coverings of the Stanford Bunny, with $n = 200$ points per sphere, for small core sizes.

8.2 Approximation Behaviour

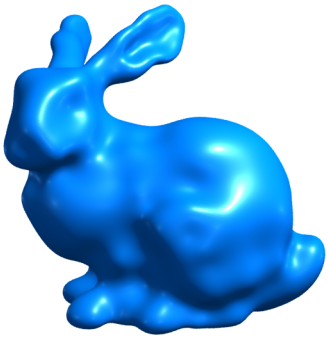
Our default primary basic function is the multiquadric which we use with a linear polynomial term, and for Late Hole Detection extra spheres we use the Wendland $\Phi_{3,1}$ with a support radius 80% of the radius of the sphere. For the multiquadric, we use an automatic parameter selection method. There is a lot of computational experience that as the multiquadric parameter ν (see page 74) gets large with respect to the region of operation, the condition number gets large, but, up to a point, the computations work well for smooth functions. See for example Tarwater [112] as reported by Kansa [56]. A folklore result is that setting ν to be the expected minimum distance between points works well. We estimate this by assuming that the surface is a plane going through the centre of a sphere with radius r and that the points are arranged on a square grid, leading to $\nu = \sqrt{\pi r^2/n}$.

Figure 8.4 shows a three level fit to the Stanford Bunny test dataset, using both Late Hole Detection (see Section 7.6, page 82) and Level-Aware Partition of Unity (see Section 7.7, page 89), and based on a common base level greedy fit. As both methods use the same sphere covering (before extra spheres) and the same multiquadric basic functions, the fits are very similar, and almost indistinguishable to the naked eye (see the tip of the near ear, especially in the middle level). These were calculated with $c = 35\%$, $n = 100$, and $\omega_{\mathrm{L}} = 1$, and all fits in this section are Hermite.

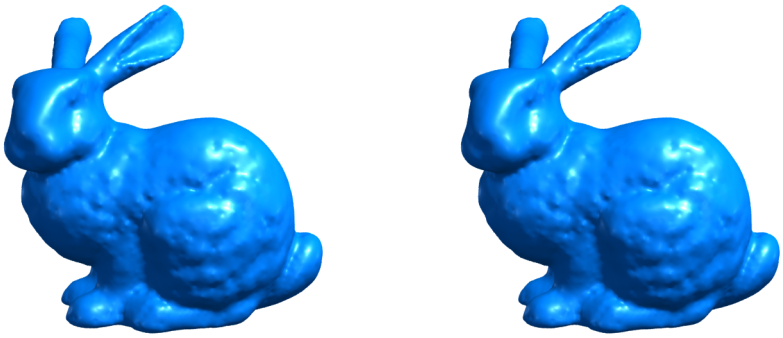
The two methods both interpolate, but differ between points due to the influence of the level below in the LAPOU case, which is greater near the edges of spheres not covered by other spheres. The primary difference in behaviour between LAPOU and LHD is in the transitions around filled holes, which are not visible in Figure 8.4, but can be seen in the views of the same fits from below in Figure 8.5.

As expected, the base level global fit easily fills the four holes on the bottom, and then the upper two levels refine the model around around them. These holes are quite small, and as shown in Figure 8.6, the middle level spheres are comfortably big enough to cover them easily. The top level spheres are not, however, and LHD adds three extra spheres (red) to fill the gaps. In the views from below of the top level spheres, the effects of the holes' presence can clearly be seen. Around the circular holes the spheres get larger, since on one side they are reaching out into empty space. The strip holes on the edges of the inset section also make a difference: the spheres along the top edge are larger than those on the opposite edge, as are the spheres on the lower right of the left circular hole compared with those on the upper right.

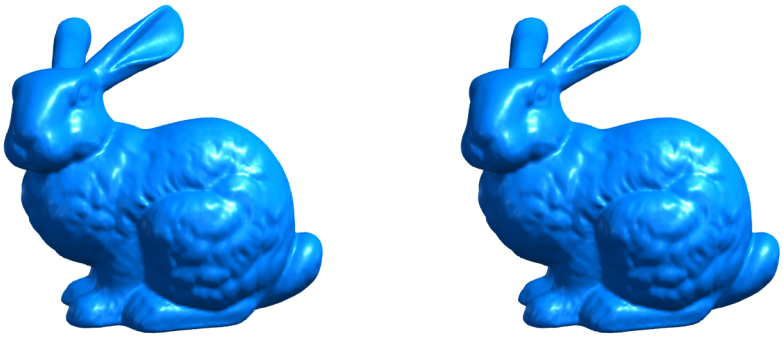
In order to demonstrate hole filling more dramatically, we modify the Stan-



(a) Greedy base level fit using 250 points and normals.

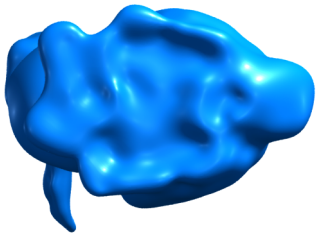


(b) Middle level piecewise fit to approximately 10% of the points

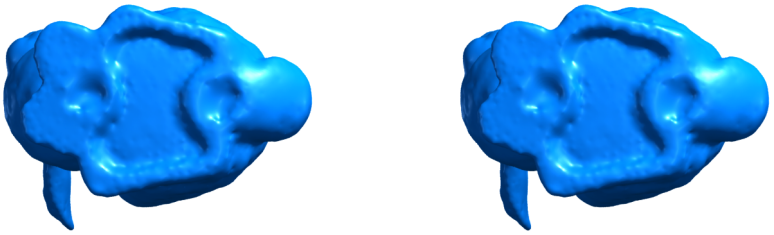


(c) Top level piecewise fit to all points

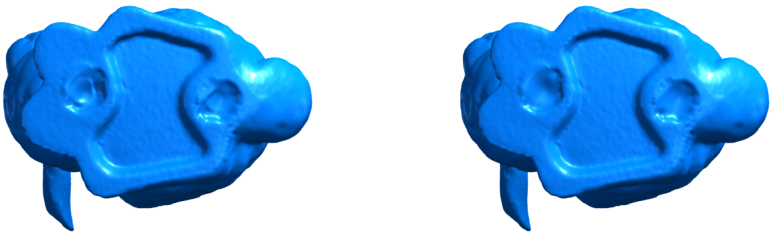
Figure 8.4: A three level reconstruction, with Late Hole Detection on the left and Level-Aware Partition of Unity on the right. Each level is a refinement of the level below.



(a) Greedy base level fit using 250 points and normals.

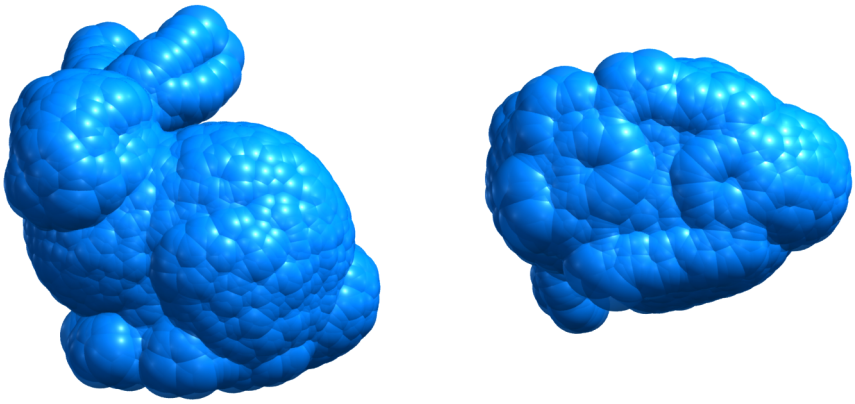


(b) Middle level piecewise fit to approximately 10% of the points

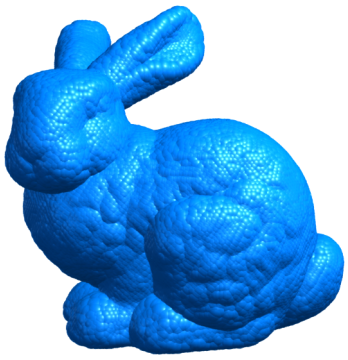


(c) Top level piecewise fit to all points

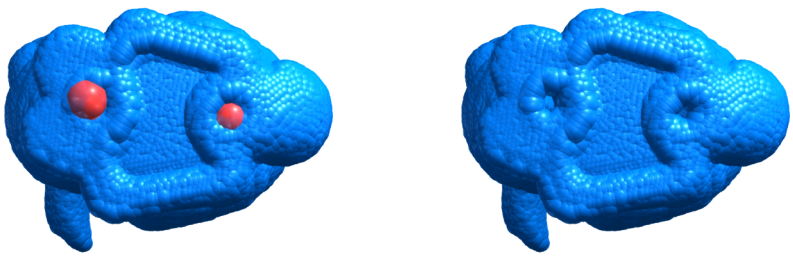
Figure 8.5: The same three level reconstruction as in Figure 8.4, viewed from below. As before, LHD is on the left and LAPOU is on the right.



(a) Middle level spheres for both LHD and LAPOU, from the side (left), and from below (right). 1,168 ordinary spheres, minimum, average and maximum radius 0.063, 0.082 and 0.10 respectively.



(b) Top level spheres for both LHD and LAPOU (from this view they are identical). 12,148 ordinary spheres, minimum, average and maximum radius 0.024, 0.029 and 0.043 respectively.



(c) Top level spheres from below for LHD (left), and LAPOU (right).

Figure 8.6: Spheres used in the multilevel fits of Figures 8.4 and 8.5.

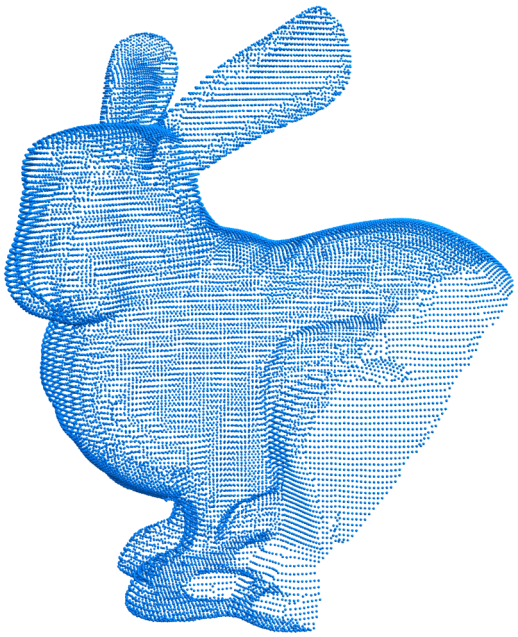
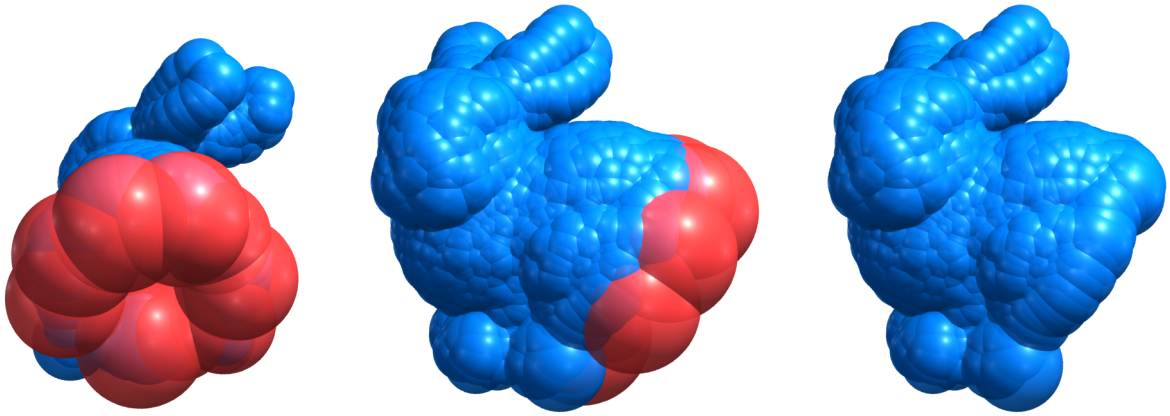


Figure 8.7: The points of the Stanford Bunny test dataset with a large hole.

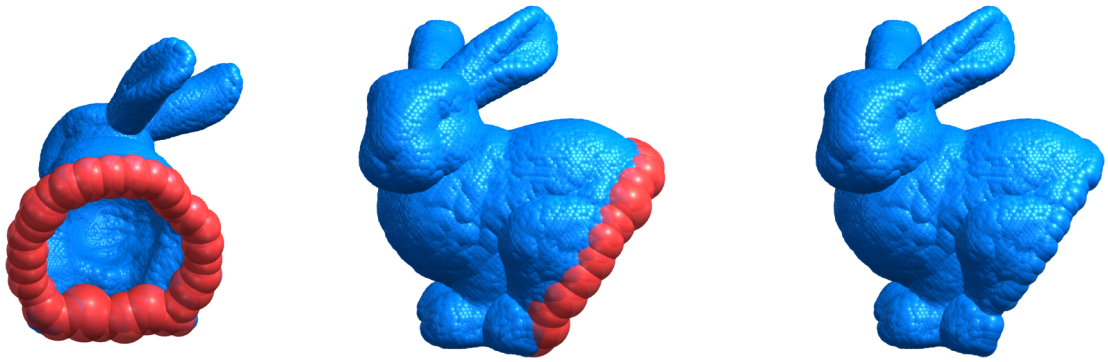
ford Bunny test dataset as shown in Figure 8.7, removing all points inside the sphere of radius 0.6 centred at $(1, 0, 0.5)$ and leaving 24,423 points. This makes a much larger hole, a size that could arise, for example, from a scan of a damaged object or an incomplete scan.

The sphere coverings generated by the same kind of piecewise fits as before are shown in Figure 8.8, and now LHD produces many more extra spheres. The fits themselves, shown in Figure 8.9 remain very similar to each other. The global base level again fills the hole (perhaps somewhat surprisingly, given its size), and the higher levels blend into it with their respective strategies. Note the large size of the spheres around the edge of the hole; they have to expand more to cover their n points, and the LHD extra spheres are centred out into the hole a little as well. The LHD reconstruction was polygonised on a $2^8 \times 2^8 \times 2^8$ grid, producing 609,110 triangles from a total of 498,673 evaluations, of which 148,521 were invalidated by a new sphere. The LAPOU lower level weight ω_{L} is 1 for these figures, and two other values are shown in Figure 8.10.

These surface reconstruction methods have a number of parameters, and it can be difficult to determine which values they should be given, and what the trade-offs might be. While some parameters have intuitively predictable

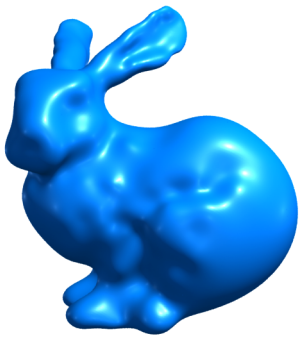


(a) Middle level spheres for LHD, left and centre, and LAPOU, right. LHD added 9 extra spheres.

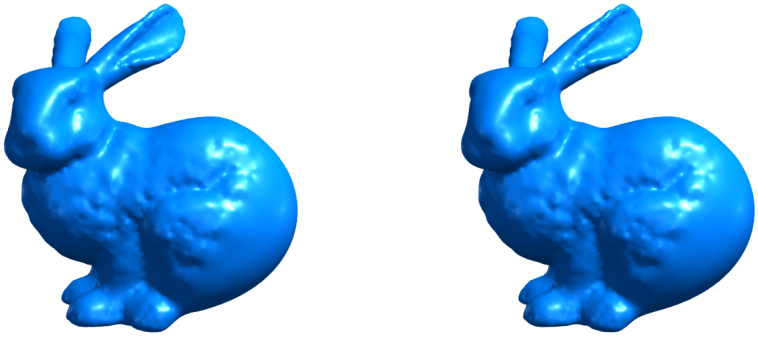


(b) Top level spheres for LHD, left and centre, and LAPOU, right. LHD added 33 extra spheres.

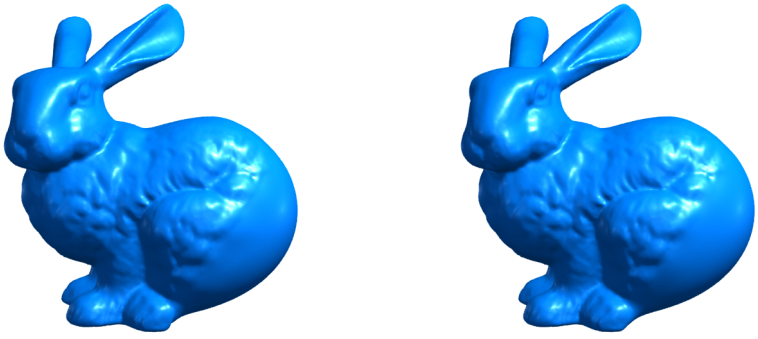
Figure 8.8: Spheres used in the multilevel fits of Figure 8.9.



(a) Greedy base level fit using 250 points and normals.

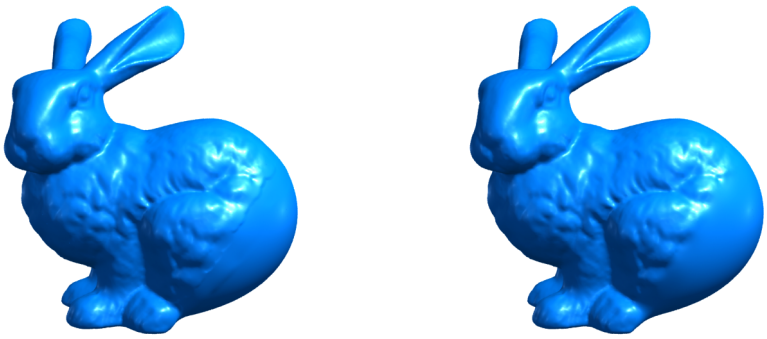


(b) Middle level piecewise fit to approximately 10% of the points.



(c) Top level piecewise fit to all points.

Figure 8.9: Three level reconstruction, with LHD on the left and LAPOU on the right.



(a) $\omega_L = 0.005$. (b) $\omega_L = 5$.

Figure 8.10: Hole filling with LAPOU using two different weights for the level below. Note the abrupt transition to the smoother hole-filling surface in the left image.

responses, n (the number of points per sphere) and c (the core proportion) and their interaction have proven particularly tricky to get a handle on. We have performed an extensive range of tests to determine which parameter values are best for various performance goals.

The two primary measures of performance are the time taken and the accuracy of the end result. To measure time, we take the best fit time from five well-separated runs; a large proportion of the fit time is spent in evaluating the level below, so we consider polygonisation to be represented well by this also. To measure accuracy, we carry out a multilevel fit to half the points of a dataset and look at the absolute value of the resulting function at the data points, the ℓ_1 error in the function value. The test machine we use has 8 Intel Xeon X5482 CPU cores running at 3.20 gigahertz, and 32 gigabytes of RAM. The Operating System is the x86_64 edition of openSUSE 11.1, with Linux kernel 2.6.27.7. These parameter tests run on a single core and are never in danger of running out of memory.

We performed the parameter tests using both the Stanford Bunny and the Stanford Dragon datasets, and the results were similar but not identical. As our goal is to find appropriate parameter values to use with other larger datasets where such extensive testing is not feasible, we consider the Dragon results to be more representative of future use, and these are the ones presented here. Also, the Dragon results are easier to understand, as they are less tightly spaced and contain less noise from timing discrepancies. The Stanford Dragon was

also produced by the Stanford University Computer Graphics Laboratory[2], and after being processed in the same way as the Bunny, described above, consists of 435,545 points and normals.

The accuracy versus time trade-off is shown in Figure 8.11. Each subfigure represents the same calculations but with a different number of centres, n_g, used for the globally supported base level fit. As the fits here are all Hermite, the base level fit consists of $n_g/2$ surface points, each with a normal direction; there are two basis functions corresponding to each centre, one pointwise and one derivative. The surface reconstruction has two piecewise levels, a middle level with 5% of the points and their associated normals, and a top level with 50% of the points and their normals, all chosen by sphere coverings as described in Section 8.1. We have found this to be a good ratio of level sizes for most models, but better level configurations will be achievable for individual datasets, and in particular, higher ratios seem to work better for larger models.

In the figure, each small circle represents a multilevel fit with one parameter combination (n_g, n, c), positioned according to the total fit time in seconds, on the horizontal axis, and the final average ℓ_1 error, on the logarithmic vertical axis. n_g varies only between subfigures, and n and c vary the same way within each subfigure. n runs from 10 points per sphere on the left to 300 on the right in steps of 10, while c runs from 20% (high overlap) at the bottom to 80% (low overlap) in the top left in steps of 5% (calculations for higher c values were also performed, but they led to higher error and are not shown). The blue lines connect points with the same c value, as indicated by arrows, and the green lines connect points with the same n value, also indicated by arrows. While the $n_g = 100$ case appears to be slower than $n_g = 250$, this is an illusion caused by its errors being much higher for short fit times.

Several general trends are apparent:

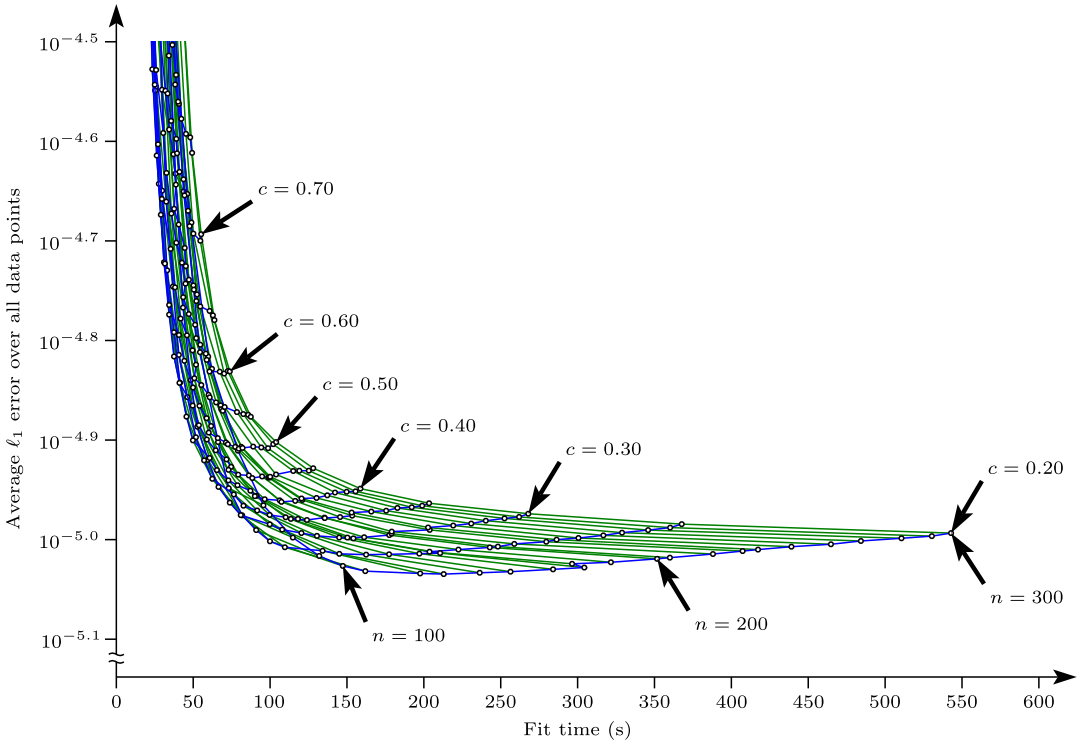
- Fit times increase with more points per sphere, especially with greater overlap (more spheres).
- Error rises dramatically for lower numbers of points per sphere, and for lower overlap.
- Smaller global fit sizes lead to lower best possible errors (surprisingly), but the lowest overall error by this measure lies in $100 < n_g < 500$.

In our experience, fewer than 500 base level centres will rarely be enough for filling large holes satisfactorily, so we choose $n_g = 500$ as our default. From the $n_g = 500$ plot we choose $n = 100$ and $c = 35\%$ to give a good combination

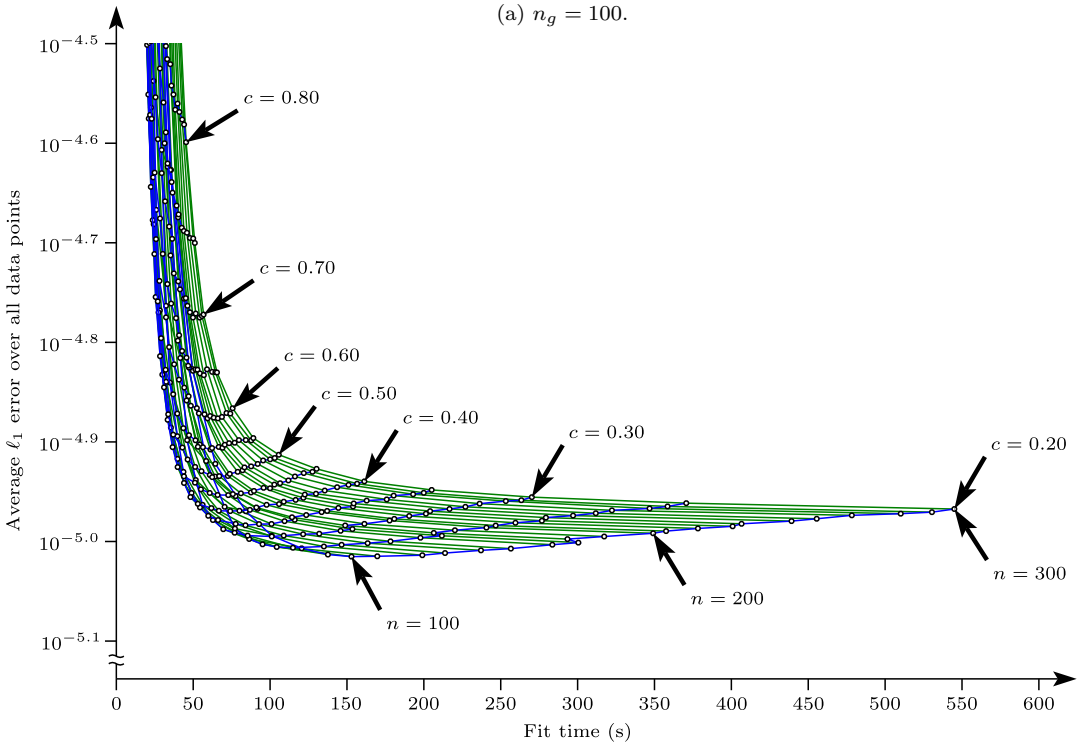
[2]From the package `dragon_recon.tar.gz`, available from the Stanford 3D Scanning Repository: `http://graphics.stanford.edu/data/3Dscanrep/`.

of speed and accuracy. The absolute best combination of parameters and level configuration varies depending on a dataset's size, complexity, and hole filling requirements, but there will be a wide region in parameter space which will work well.

One further important consideration is the polygonisation grid size. If it is too large relative to the size of the spheres, the polygoniser may step outside the spheres, or between them if overlap is small, with a step large enough that the blend region of LAPOU is skipped. The polygoniser can then leap across onto disconnected blobby pieces left over from the coarser lower levels of the approximation. This problem is generally worse in Hermite fits; the off-surface points added in a pointwise fit, if they are far enough out, may mean there are spheres at the top level reaching further away from the surface. Hermite fits also have fewer centres to work with. The issue can be avoided by ensuring a good match between the number of data points used and the detail of the polygonisation, thinning the dataset or decreasing the polygonisation grid size as necessary, though the other parameters of the method can have a useful effect as well. Reducing ω_{L} is often beneficial in situations of mismatched sphere and grid sizes.

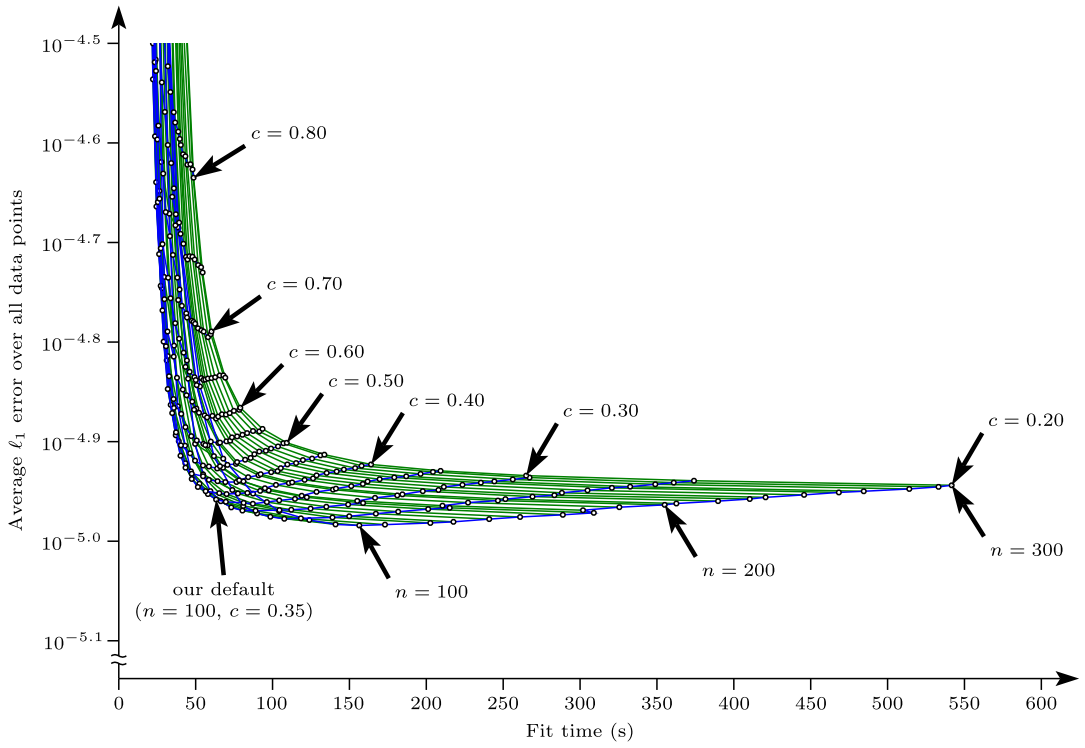


(a) $n_g = 100$.

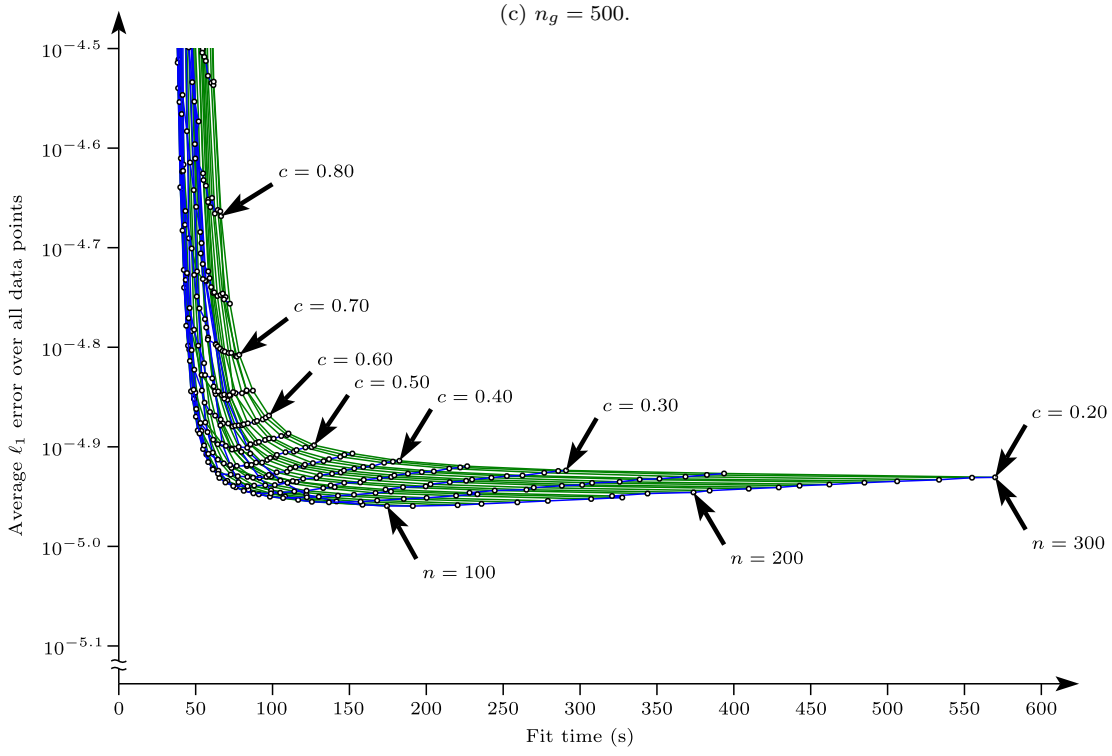


(b) $n_g = 250$.

Figure 8.11: Accuracy versus fit time for the Stanford Dragon dataset over a wide range of parameter values.



(c) $n_g = 500$.



(d) $n_g = 1000$.

Figure 8.11: Accuracy versus fit time continued.

8.3 Scalability

In this section we present results for different, larger datasets. We are particularly interested in scalability, the way computational requirements (especially time) increase as dataset size increases. For example, a more efficient method will be able to handle, say, a million points using fewer resources, while a more scalable method will have a low rate of resource requirements growth as dataset size increases, that is, fewer resources *per* million points.

All fits here use LAPOU, but we compare Hermite to pointwise in terms of speed and lower level appearance. The top level Hermite and pointwise reconstructions are visually identical except for the shapes with which holes are filled; that difference is as much a result of different sphere positions as anything else. All times quoted are the best of five runs.

Timings for the Stanford Bunny as reconstructed above and the pointwise equivalent (two off-surface points per surface point, at distance 0.001) are shown in Figure 8.12. These were calculated with core proportion $c = 35\%$, number of points per sphere $n = 100$, and level blending weight $\omega_{\mathrm{L}} = 1$, and polygonised on a $2^9 \times 2^9 \times 2^9$ grid. In the figure, the subtasks are "residuals", evaluating the approximation built up by the previous levels at this level's points; "spheres", building a point tree and constructing a sphere covering; "local fits", performing the local approximations in the spheres; and "sphere tree", building a sphere tree for evaluation. Note that the pointwise method is faster for a model of this size, despite having half again as many interpolation conditions.

Hermite results for the Stanford Dragon are shown in Figures 8.13 and 8.14. These were calculated with $c = 35\%$, $n = 100$, and $\omega_{\mathrm{L}} = 0.05$, and polygonised on a $2^9 \times 2^9 \times 2^9$ grid (almost too coarse for these sphere sizes). Timing results for this reconstruction and for the pointwise equivalent (with off-surface distance 0.001) are shown in Figure 8.15. The pointwise fit is slower for this dataset.

We move next to a much larger model, the Stanford Lucy, also produced by the Stanford University Computer Graphics Laboratory[3]. We processed the model in the same way as the Bunny and Dragon, removed two duplicated points[4] and a few extraneous points near the tip of one finger[5], and were left with 14,027,865 points and normals.

Hermite and pointwise results for this model are shown in Figures 8.16 and 8.17. These were calculated with $c = 35\%$, $n = 100$, and $\omega_{\mathrm{L}} = 0.05$ for the middle level, but we found it helpful for both speed and polygonisation to increase the size of the spheres with $n = 180$ and increase their spacing

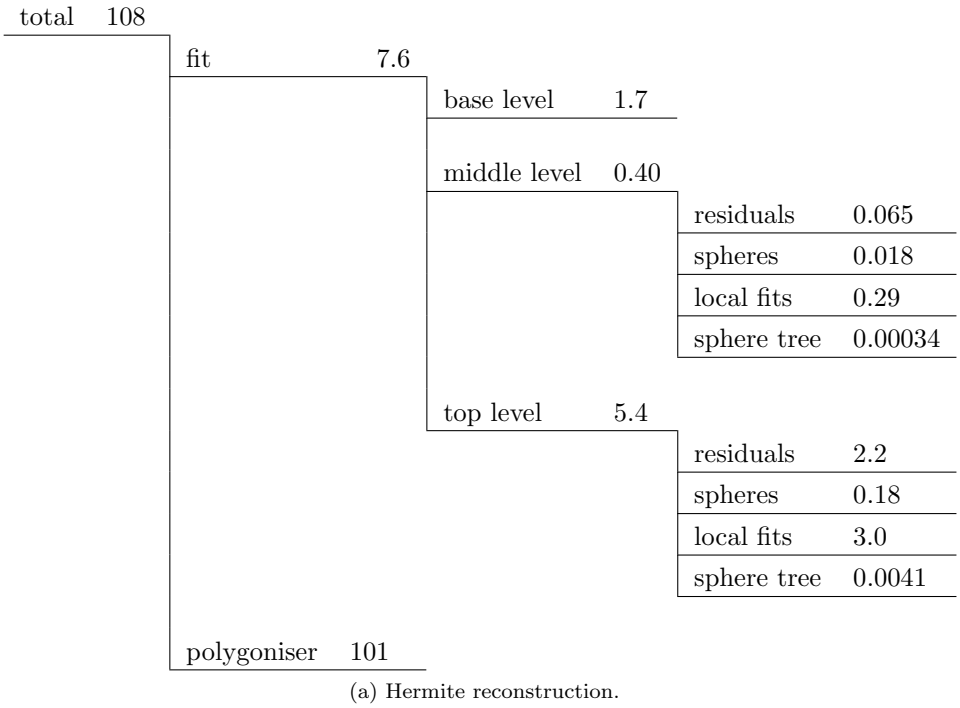
[3]From the package `lucy.tar.gz`, available from the Stanford 3D Scanning Repository: `http://graphics.stanford.edu/data/3Dscanrep/`.

[4]Numbers 2,855,827 and 10,896,371 (starting from 0).

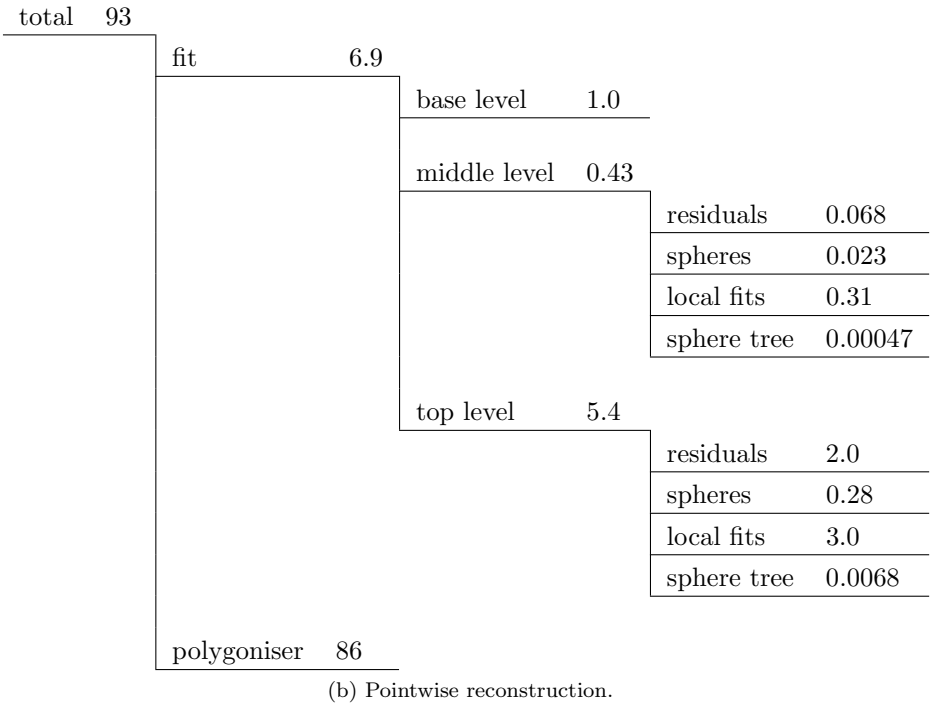
[5]Numbers 6,184,677, 6,184,699, 6,187,892, 6,187,893, 13,782,055.

with $c = 50\%$ for the top level. The pointwise fits used an off-surface distance of 0.0001. The fits were polygonised on grids of sizes $1/2^9$, $1/2^{10}$, and $1/2^{12}$ for the base, middle and top level figures. $1/2^{12}$ works well for the pointwise version, but it is still too large for the Hermite fit.

Timing results for these reconstructions are shown in Figure 8.18. The pointwise fit is again slower. These times do not include polygonisation, and it is a significant cost at several tens of hours. A Hermite fit polygonised on a $2^{12} \times 2^{12} \times 2^{12}$ grid took around 60 hours, but this includes the time taken to polygonise some large blobby artefacts. The pointwise fit shown in Figure 8.16e was computed using block-based polygonisation and on-the-fly fitting by four separate instances of the same serial program running simultaneously on two different machines. Times for the blocks varied between 21 and 27 hours, but note that neither the blocks nor the machines were equal, and that some computation was repeated in the independent processes. A full Hermite fit (not on-the-fly) takes around 18GB of memory, and we discuss the reasons for this large size later in this section.

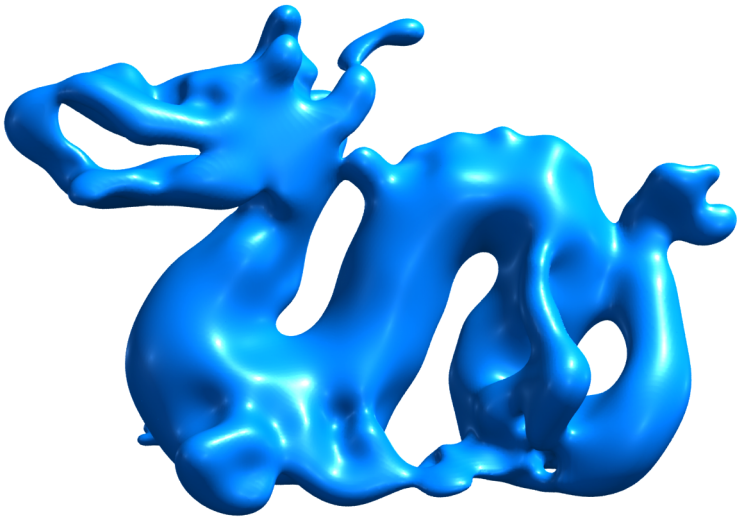


(a) Hermite reconstruction.

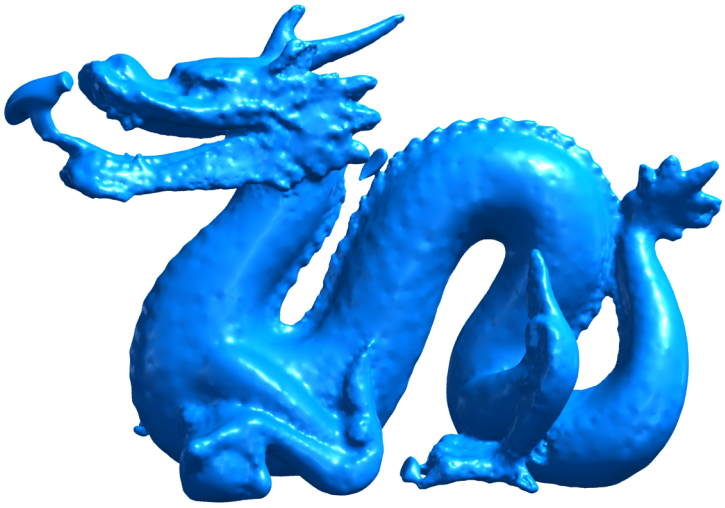


(b) Pointwise reconstruction.

Figure 8.12: Breakdowns of the time taken by the main stages in the reconstruction process for the Stanford Bunny. The numbers are times in seconds and do not add up precisely due to rounding and omitted minor stages.

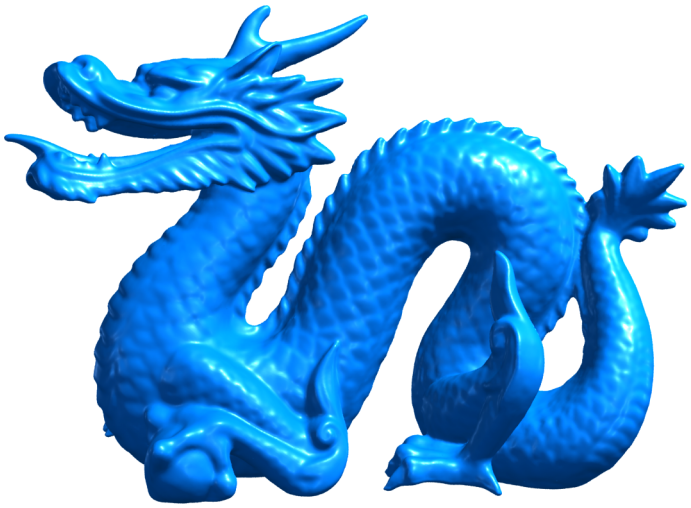


(a) Greedy base level fit using 250 points and normals.



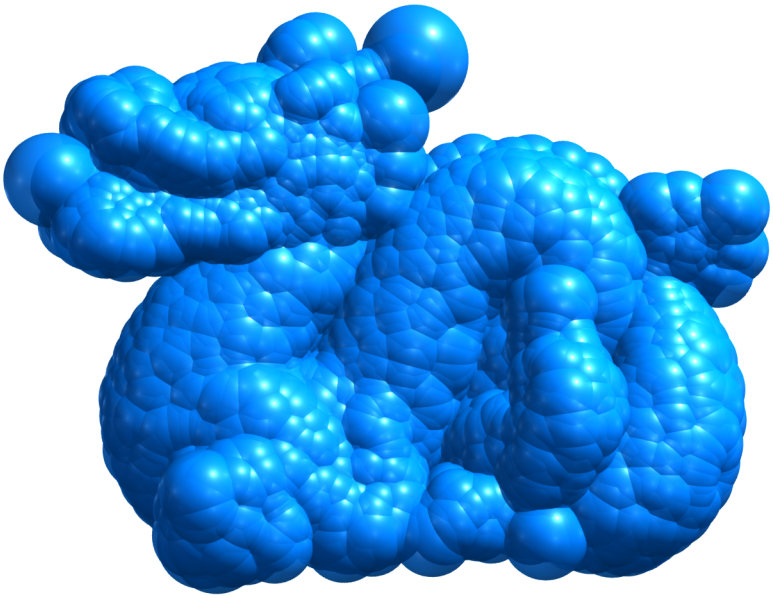
(b) Middle level fit to 2% of the data. Note the extra blobs still present.

Figure 8.13: Multilevel Hermite reconstruction of the Stanford Dragon.

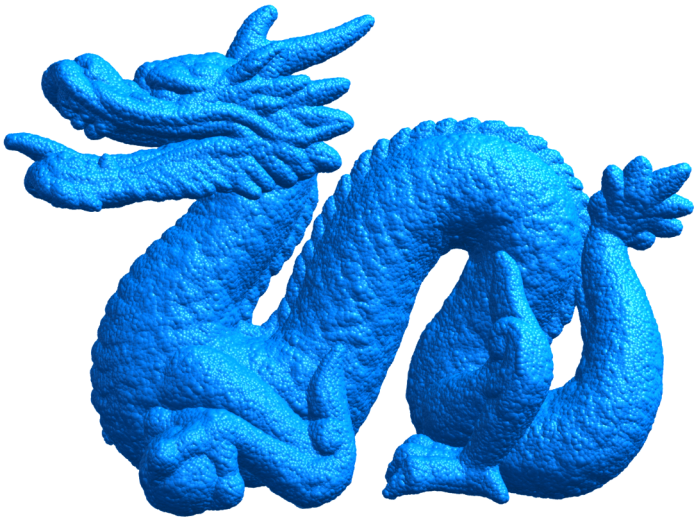


(c) Top level fit using all data. All blobs have now been cleared away.

Figure 8.13: Stanford Dragon reconstruction continued.

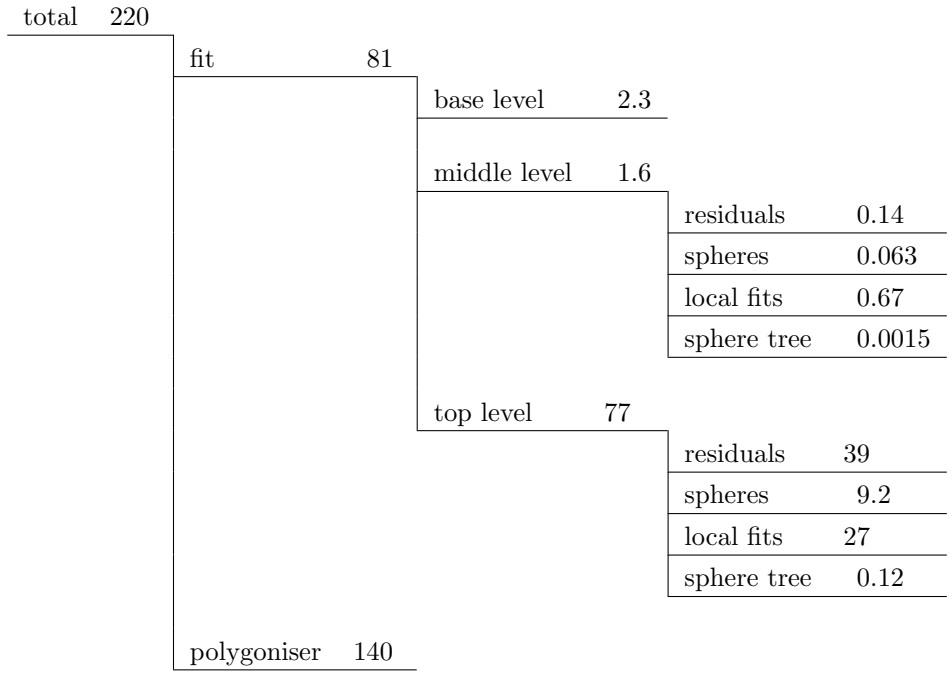


(a) Middle level, 2,720 spheres, with minimum, average and maximum radius 0.026, 0.044 and 0.075 respectively.

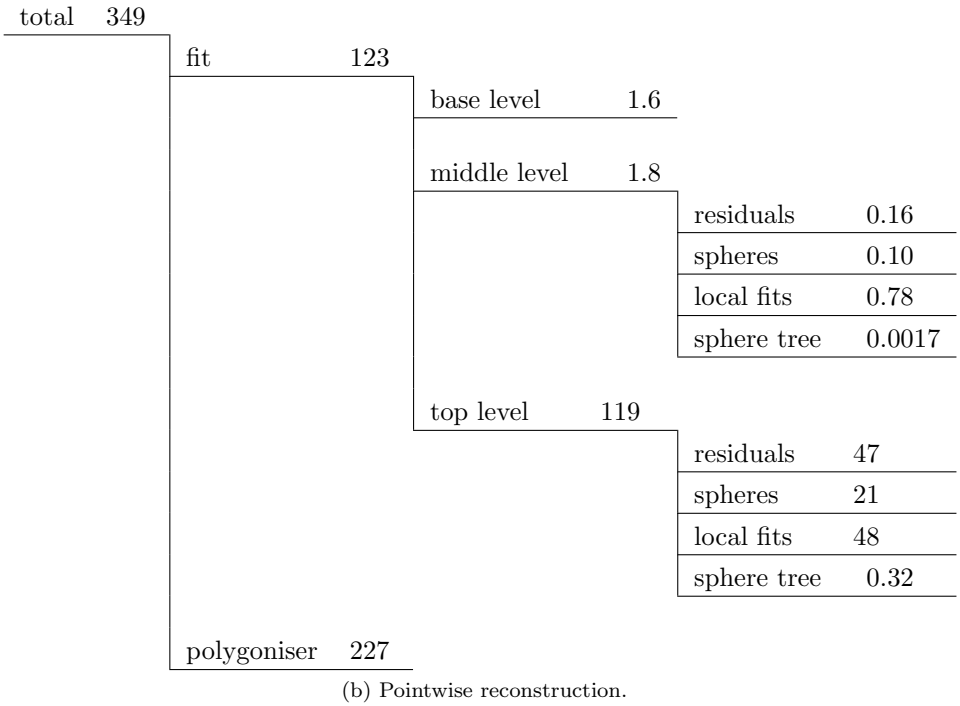


(b) Top level, 109,271 spheres, with minimum, average and maximum radius 0.0012, 0.0068 and 0.020 respectively.

Figure 8.14: Spheres associated with the middle and top levels of the Stanford Dragon reconstruction in Figure 8.13.

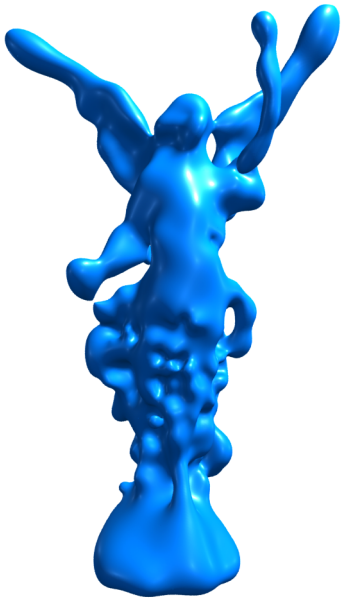


(a) Hermite reconstruction.

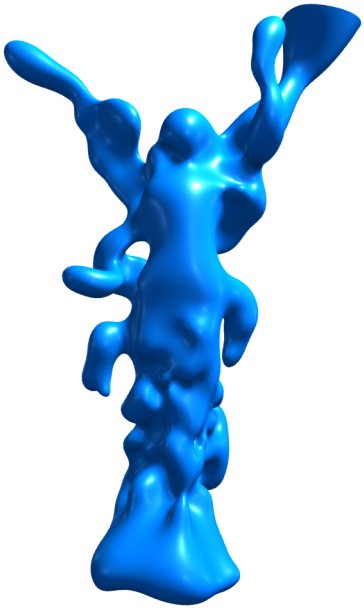


(b) Pointwise reconstruction.

Figure 8.15: Breakdowns of the time taken by the main stages in the reconstruction process for the Stanford dragon. The numbers are times in seconds and do not add up precisely due to rounding and omitted minor stages.



(a) Hermite greedy base level fit using 250 points and normals.



(b) Pointwise greedy base level fit using 166 surface points, each with an off-surface pair.

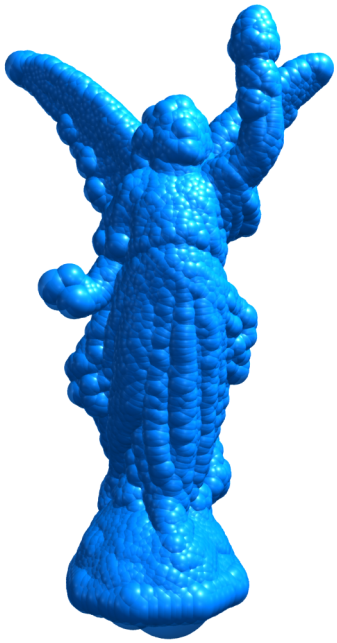
(c) Hermite middle level fit to 0.1% of the data.

(d) Pointwise middle level fit to 0.1% of the data.

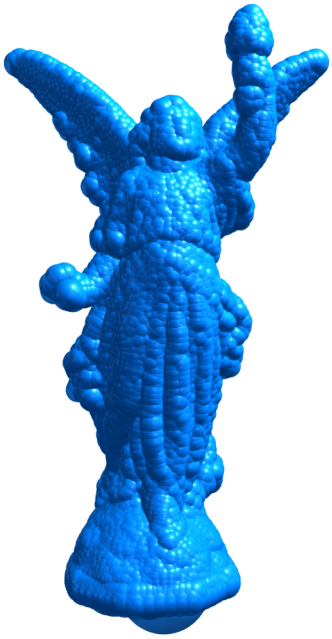
Figure 8.16: Multilevel reconstruction of the Stanford Lucy. Extra blobs are still present on the middle level fits.

(e) Pointwise top level fit using all data. For this method, all blobs have now been cleared away.

Figure 8.16: Stanford Lucy reconstruction continued.



(a) Hermite middle level, 6,856 spheres, with minimum, average and maximum radius 0.014, 0.021 and 0.078 respectively.



(b) Pointwise middle level, 10,199 spheres, with minimum, average and maximum radius 0.012, 0.017 and 0.077 respectively.

(c) Hermite top level, 1,488,113 spheres, with minimum, average and maximum radius 0.00095, 0.0012 and 0.048 respectively.

(d) Pointwise top level, 2,040,593 spheres, with minimum, average and maximum radius 0.00076, 0.0010 and 0.040 respectively.

Figure 8.17: Spheres associated with the Stanford Lucy reconstruction in Figure 8.16. Note the extra-large spheres visible at the bottom of the middle level figures; the dataset has a very low density of points underneath the statue, which gives rise to the unexpectedly large maximum sphere sizes.

fit	2601		
	base level	12	
	middle level	13	
		residuals	0.34
		spheres	0.15
		local fits	1.7
		sphere tree	0.0034
	top level	2573	
		residuals	1209
		spheres	62
		local fits	1293
		sphere tree	1.5

(a) Hermite reconstruction.

fit	2733		
	base level	11	
	middle level	13	
		residuals	0.37
		spheres	0.21
		local fits	2.0
		sphere tree	0.0038
	top level	2706	
		residuals	1140
		spheres	96
		local fits	1454
		sphere tree	2.2

(b) Pointwise reconstruction.

Figure 8.18: Breakdowns of the time taken by the main stages in the reconstruction process for the Stanford Lucy. The numbers are times in seconds and do not add up precisely due to rounding and omitted minor stages.

Finally, we compare our method to the multi-scale compactly supported RBF method of Ohtake, Belyaev and Seidel [85], "multi-CSRBF". This is a multilevel modified Shepard's method using local quadric approximations, where each level also has a refining compactly supported "normalised" RBF which interpolates at the centres of the subdomains (data points, at the top level). As the approximations at each level are simpler, the method is used with many more levels, e.g. ten in place of our three.

We use the original source code[6] with very minor modifications made in order to be able to compile it with Microsoft Visual Studio 2005, to display more detailed statistics, and to easily set the size of the polygonisation grid to be the same as our method. We run this Windows executable on our Linux test machine using Wine 1.1.9. Tests on two machines with identical hardware, one running Microsoft Windows Server 2003, and one running openSUSE Linux, showed there is no performance penalty when running this software under Wine, and in fact the program ran slightly faster on Linux. As this implementation is 32-bit, even with the `/LARGEADDRESSAWARE` compiler flag it can only access 3 gigabytes of memory. This limits the size of the datasets which can be processed, and the resolution of the meshes which can be produced.

We run the multi-CSRBF software with its default parameters (no smoothing, automatically chosen number of levels, Bloomenthal polygonisation) on the same processed datasets already described. A summary of statistics for our methods and for multi-CSRBF is given in Table 8.1. Multi-CSRBF is faster for the very small Bunny model, but by the time we move to the Dragon it is already much slower, though the polygonisation stage is still very fast. This polygonisation speed is probably due to the simpler local approximations, as the two different polygonisations use a similar number of evaluations (though our implementation does produce nearly twice as many triangles). Our methods use more memory for several reasons, some due to the technique and some due to implementation choices. Our local RBF approximations take up more space than the quadrics of multi-CSRBF, especially since we store a copy of each sphere's points with the sphere for the speed benefit of having them all in the CPU's cache at once (this decision costs about 12GB for the full Hermite Lucy fit). We use double precision floating point numbers exclusively, while the implementation of multi-CSRBF uses single precision for many purposes, including storing data points. Also, multi-CSRBF does not need a sphere tree, as its spheres have a constant size on each level and a standard point tree can be used instead.

Multi-CSRBF reconstructs surfaces well for hole-free models, and the re-

[6] From the package `multi-csrbf-gui.zip`, available from Yutaka Ohtake's software page: `http://www.den.rcast.u-tokyo.ac.jp/~yu-ohtake/software/index.html`

	Bunny			Dragon		
	fit	poly.	mem.	fit	poly.	mem.
Hermite	7.6s	101s	319MB	81s	140s	860MB
pointwise	6.9s	86s	322MB	123s	227s	1289MB
multi-CSRBF	2.2s	36s	262MB	810s	47s	441MB

Table 8.1: Fit time, polygonisation time and maximum memory usage of our two methods and multi-CSRBF.

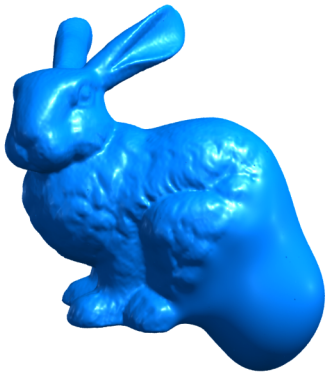


Figure 8.19: The multi-CSRBF method applied to the partial Stanford Bunny dataset as shown in Figure 8.7, page 106.

sults, not shown, are visually very similar to those of our methods. We applied multi-CSRBF to our modified Stanford Bunny with the large hole (Figure 8.7, page 106), and as Figure 8.19 shows, while the hole was filled, the result was not satisfactory. Our global base level approximation provides an advantage for this type of problem.

To observe the scalability of the methods, we performed fits to subsets of the Stanford Lucy dataset, using the parameters described above. The results are shown in Figure 8.20. Multi-CSRBF fits to larger subsets than those shown were not possible as the implementation reached its memory limit; for the full dataset this took around three hours, by which time the program had not yet reached its final level (by far the most expensive).

Both our methods scale at about the same slow rate of just over three minutes per million surface points, and the Hermite version in particular is very linear. The pointwise method is consistently slower for datasets of this size. Multi-CSRBF scales at a much steeper rate, which appears to be worsening as the number of points increases.

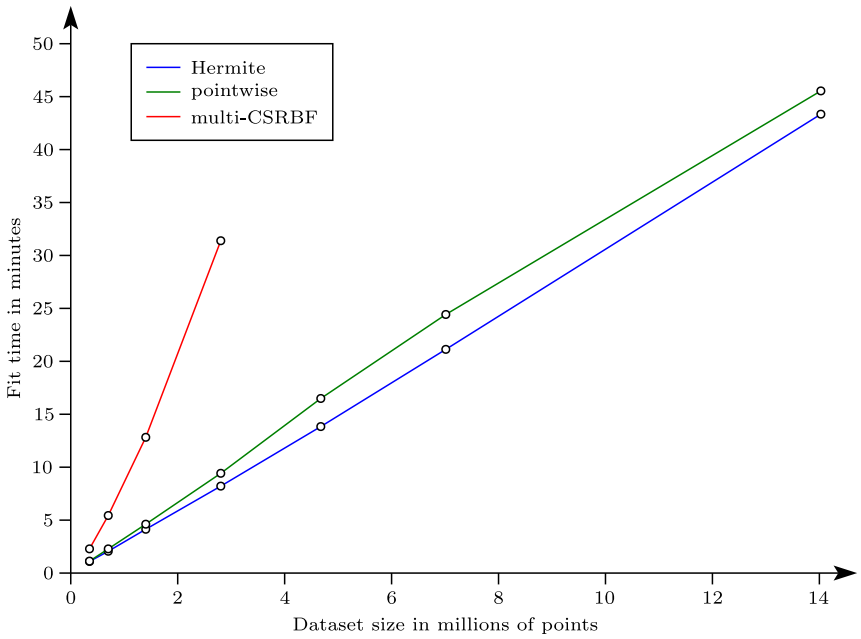


Figure 8.20: Scaling behaviour of the fit time for the Stanford Lucy and evenly spaced subsets. Hermite is in blue, and pointwise is slightly slower, in green. Multi-CSRBF, in red, exhibits much poorer scalability.

8.4 Future Directions

We have demonstrated that our sphere-based piecewise surface reconstruction method works well and scales well, but there remain many questions to answer and improvements to make.

- Using approximation rather than interpolation is essential for real-world datasets. Our purpose in this project has been to develop a reconstruction method which scales well while retaining useful features such as good hole filling, and interpolation has been perfectly suitable for this goal. We can comfortably interpolate to 14 million surface points, and using an approximating RBF with that many centres we should be able to handle many times more.

- Parallelism is another feature which has been deliberately omitted from the scope of this project, while remaining in the back of our minds as an important future goal. Potential parallelisation approaches were discussed in Section 7.8.3, page 93.

- Mismatches between sphere and polygonisation grid sizes have emerged as an important issue, in part due to our choice of such a simple polygonisation method. This is, however, representative of the conflict that exists between our desire to have an easily visualisable polygon model, and our desire to make use of the full detail of larger datasets. One potentially easy solution is to have all spheres on a level have a fixed physical size, with the size on the top level related to our intended polygonisation grid size. This would be best used in conjunction with an approximating method.

- Several authors have achieved success with partition of unity methods involving simpler approximations, such as quadratics. Our basic system and our implementation are not restricted to RBFs, and it would be interesting to try approximating with simpler functions in a subdomain before deciding to fit an RBF. An experiment which only used planes produced an interesting rounded Voronoi effect when applied to a sphere dataset.

- Several aspects of the method could benefit from more analysis. More exploration is needed into the approximation behaviour with different numbers of points per level and different numbers of levels. Our apparent relatively slow evaluation speed requires examination. The comparative strengths and weaknesses of Hermite and pointwise RBFs for this application are not yet fully understood. Mid-range points as described in Section 7.3 deserve further investigation.

Part III

Animation Control

Chapter 9

Introduction

Part III of this thesis concerns a project on which we spent a considerable amount of time, but with which we ultimately did not achieve success. This project attempts to apply Radial Basis Function approximation to the computer animation technique of motion synthesis, and in particular to the animation control problem, in order to reduce its computational requirements. This part of the thesis summarises our experience in this area, comprising relevant background material and several mathematical steps towards our goal.

9.1 Computer Animation and Motion Synthesis

Computer animation is a versatile class of methods for creating animations, movies and visual effects, to be used stand-alone or to be combined with traditionally filmed material. In computer animation, individual frames are produced automatically by a computer system from a scene description which is numerical (at least under the surface). The objects in the scene generally represent real-world objects, and their motion can be defined manually or automatically in various ways.

Our interest here is in the automatic generation of motion for articulated characters. We are not concerned with the frame rendering or the final animation, but the behind the scenes mathematical techniques for producing motion.

9.1.1 Computer Animation Methods

Computer animation can be categorised into two basic approaches, kinematics and dynamics. In kinematics, the positions and velocities of objects are specified directly but with little more regard for physical accuracy than what a skilled animator can impart or what can be transferred from a motion capture actor. In

dynamics, the physics of the situation are modelled mathematically to determine the motion, improving the physical realism at the cost of control.

Motion capture can be used to improve the physical realism of character movement by recording the actual motion of an actor, and then using this kinematic data to drive the character's animation. The physical accuracy decreases as the difference between the actor and virtual character increases in terms of size, proportion, and mass distribution, as well as when separately captured sections of motion are blended together.

The trade-off between control and physical accuracy has meant that dynamics has been used for unactuated environmental effects such as fluids, rigid bodies colliding and falling, hair and cloth, while kinematics has held sway over character animation. One entry by dynamics into character motion is with "rag-doll" physics, allowing a limp virtual stunt-person to fall realistically.

With rag-doll physics, a character is represented by an articulated rigid body model, where hinges and ball joints connect rigid pieces. It is important to note that this movement is uncontrolled—no muscles or artificial alternatives are exerting force. Since the late 1980s, however, researchers have been attempting to bring dynamics into the world of character animation by outfitting articulated rigid body models with actuators and that crucial but mostly elusive element, a control system.

9.1.2 The Animation Control Problem

An ideal held by many researchers is a system where a character (model) can be given a fairly general instruction (e.g. "walk across the room" or "catch this ball") and a controller will be automatically selected or generated that can achieve the desired result when fed into the model in a physics simulator. Creating these controllers remains extremely difficult. Controllers have been automatically generated for simple models in two and three dimensions by a variety of methods including constrained optimisation [130], genetic algorithms [80, 34, 7], sensor-actuator networks [119, 121], genetic programming [39, 40], and neural networks [107]. Success for models as complex as a very stylised humanoid has required manual intervention such as careful sensor selection [107], a hand-specified state sequence [62] or a supporting "hand of God" [122]. Controllers for somewhat more realistic human models have been hand-created with significant effort, and require careful analysis of specific motions [45, 44, 27, 26, 138]. A wide variety of solutions have been attempted for this problem and our review here is by no means complete. In particular, a great deal of research has focused on finding appropriate controller representations and optimisation techniques. We are concentrating on a lower level part of the problem here.

Each of the automatic controller generators is at its heart an optimisation process minimising a motion-judging evaluation metric over a multidimensional space of possible controllers. This optimisation is made difficult by high dimensionality, lack of gradient information and a high cost of evaluating the motion—each trial requires calling the physics simulator, which runs at best only a small factor faster than realtime.

This research addresses the evaluation cost and gradient problems by introducing an approximating cache layer between the optimiser and the physics simulator. The approximation can represent huge time steps compared to the tiny steps physical simulation often requires for stability, and can be continuous and differentiable.

9.2 Outline

In Chapter 10 we review the types of physical simulation used in motion synthesis, including several types of actuator and their desirable and undesirable features.

In Chapter 11 we set up a Radial Basis Function scheme for approximating simulation results, solve some computational issues, and discuss the limitations of this approach.

Chapter 10

Simulation

10.1 Physics Modelling

We are interested in simulating models constructed of rigid segments connected by hinge joints. For this particular project, we are not interested in collisions, friction, or articulated models containing loops. The open source library ODE [106] (Open Dynamics Engine), based originally on software written by Russell Smith as part of his PhD research on motion synthesis [107], fits these needs well, and more besides.

In ODE, rigid bodies are given an inertia tensor to specify mass distribution and a shape for visualisation and collision detection; we restrict ourselves to (uniform density) cylinders for both. Rigid bodies are connected to each other and their world by joints such as hinges, ball joints and sliders; we use only hinges. The simulation consists of numerical integration of the equations of motion, and uses a semi-implicit first order integration scheme which is very stable but has low accuracy for long simulations or large integration step sizes. It is perfectly adequate for our purposes, however, as the entire process is deterministic and we will be simulating only short sections of motion from precise start points. We can essentially choose the integration steps to be as small as we need, since our primary objective is to avoid using the simulator whenever possible.

ODE represents the status of each body in "maximal" coordinates, that is, world-relative coordinates which can represent arbitrary translations and rotations, and as a consequence joint constraints are not hard. Errors can creep in, causing joint constraints to be violated, and ODE has an error reduction mechanism which applies forces to correct constraint violations. The errors, like the integration errors, do not concern us, and ODE's error reduction has proven satisfactory (we use an Error Reduction Parameter of 0.8). For our

goals of recording states and abstracting to build approximations, "generalised" coordinates will be much more practical. Generalised coordinates parameterise an articulated model by internal coordinates such as joint angles, and can hence only represent configurations which satisfy the joint constraints—a space of much lower dimension.

At each integration step, ODE accumulates the forces acting on each rigid body, and arbitrary forces and torques can be added in at this point, allowing us to plug in any actuation system we choose. We take this approach rather than make use of the built in motors as they are based on achieving a velocity around a joint, and we would like more freedom to experiment. We make use of ODE's built in gravity.

ODE provides joint limits, so that a hinge's range of motion can be restricted, but these act like solid blocks, causing bounces involving instantaneous changes in velocity. While in some applications this may be more realistic, in this project there are two reasons for it to be undesirable. Firstly, we are intending to approximate this entire system with a continuous approximation, so any discontinuities will cause trouble. Secondly, we are primarily interested in mimicking people and animals, and biological joints have cushioning and surrounding tissue which prevent such joint bounces as occur from being perfectly rigid. We make no attempt at biological realism here, but replace ODE's hard joint limits with cubic springs to ensure continuity of velocity on the time scales we are dealing with.

10.2 Actuation Modelling

10.2.1 PD Actuators

A Proportional-Derivative (PD) actuator produces an activation a (often torque or force) which tries to move a state variable x to a target position x_t. Its basic form is

$$a = k_p(x_t - x) - k_d\dot{x}, \tag{10.1}$$

where the constants k_p and k_d are the *proportional gain* and *derivative gain*, respectively. See, for example, introductory textbooks such as Hughes, *Measurement and Control Basics* [47]. PD actuators are more often referred to as PD *controllers*; we use the term "actuator" here as in our situation their parameters are in turn controlled by a higher level "controller", created by the motion synthesis process.

Note that this actuator is mathematically equivalent to a linear spring with a dashpot damper in parallel. That is, the same effect can be achieved by

connecting the object measured by x to its target position with a damped spring with rest length 0; the target velocity is implicitly 0. An alternate form aims at a specified target velocity, $\dot{x}_t$ as well:

$$a = k_p(x_t - x) + k_d(\dot{x}_t - \dot{x}). \tag{10.2}$$

More commonly used in engineering are the closely related PID (Proportional-Integral-Derivative) actuators, which have an integral term to deal with steady state error:

$$a = k_p(x_t - x) + k_i \int_0^t (x_t - x(\tau))\ d\tau - k_d\dot{x}, \tag{10.3}$$

where k_i is the *integral gain*.

The units of the control signal a depend on the units of the gain constants, and in motion synthesis a is usually joint torque or input to some torque generator for articulated rigid body models, or force for mass-spring lattice models.

Most motion synthesis research has used PD actuators, either directly, controlling joint torques or muscle activation, or indirectly, as part of the morphology of mass-spring lattice creatures. This is a little surprising, as they are not widely used in other related fields. Control theory as applied in robotics and engineering is vastly more sophisticated, and even among simple cases PID actuators are much more common than plain PD actuators. Biomechanics, of course, pays much more attention to the actual properties of biological tissue, the way muscle force varies with contraction speed and distance, and the energy-storing elasticity of tissue.

The zero-velocity form (10.1) is widely used in motion synthesis, and though a few authors have used the target-velocity form in animation (10.2), it has not generally been used for automatic motion synthesis as it doubles the dimension of the search space. In either case, the general goal is to find some kind of sequence of target positions for each joint, so that when the system is simulated, and each of the PD actuators tries to achieve its targets, the model performs some kind of desired motion.

There is an important distinction between the way PD actuators are usually used in motion synthesis and the way they are usually used in other fields. Since the independently acting actuators will interfere with each other, and since their targets may change at any time without there being any need or expectation for the previous targets to have been reached, the target positions have been abstracted into something more like control signals than targets to move to. For this reason, the steady state error correction added by PID actuators is not very important, as, should a steady state situation occur, the PD actuator targets will be chosen so that the model achieves its desired steady state position, even

if those targets differ from that position. Effectively, the joint actuators are themselves controlled at a higher level by the motion synthesis process.

10.2.2 Variable Gain PD Actuators

Despite the popularity of PD actuators in motion synthesis, they are not actually a very good model of the biological systems they are being used to simulate, and the unnatural nature of the resulting motion has been widely recognised, for example:

> "...not surprisingly, simulated articulated linkages that are regulated by simple motion controllers often move with robotic rigidity and stiffness."—Christensen, Marks and Ngo, 1995 [21]
>
> "The motions obtained to date using our technique do not yet represent convincing human motion."—Laszlo, 1996 [61]
>
> "The robotically stiff motion that has come to typify physically based approaches belies the fact that dynamics has much to offer in facilitating far more subtle motion in which animators could freely 'shape' a motion."—Neff and Fiume, 2002 [78]

Unfortunately, surprisingly little has been done to improve the situation. PD actuators have the advantage that their parameters directly correspond to easily understood physical parameters such as joint angles (though, as mentioned above, the correspondence is often more abstract in practice), which also integrate well with keyframed animation. Further, with only one parameter per actuator, the control space to be searched is easy to understand and of relatively low dimension. This control space dimension, however, is probably too low for realistic motion, as using the target angle as the only parameter means keeping the gain constant, which explicitly prevents relaxed and passive motion, and control of joint stiffness.

In their 2002 paper [78], Neff and Fiume describe this problem and introduce an antagonistic actuator, consisting of two opposed linear springs, each pulling towards a fixed target angle of their respective joint limit, but with their gains as parameters. As is pointed out in an appendix, this is mathematically equivalent to a PD actuator with both the target angle x_t and proportional gain k_p as parameters:

$$a = k_p(x_t - x) - k_d\dot{x}, \tag{10.4}$$

where k_d remains constant. What is not pointed out is that this actuator formulation (without the damping term) is the one used by Witkin and Kass in [130], one of the earliest motion synthesis papers published, a formulation which has hardly been mentioned since.

Some authors have manually made particular joints completely passive for part or all of a motion sequence, for example van de Panne and Lamouret [122] and Hodgins and Wooten [44], but the idea of making PD actuator gains into parameters in their own right has gone largely untested. Part of the reason for this is that adding an extra parameter to every actuator doubles the dimension of the controller space to be searched, making most motion synthesis schemes significantly more difficult.

10.2.3 Raw Torques

In principle, using motion synthesis techniques to find joint torques directly can achieve the behaviour of PD actuators, passive motion and everything in between, though the resolution of the torque signal may need to be quite high to capture details handled automatically by the higher level PD system. Direct torques have the advantage of having only one parameter per actuator, but the disadvantage of no longer working in terms of the easily understood physical quantities of position and tension. Several authors have used torques directly in motion synthesis, including van de Panne, Fiume and Vranesic [120], Lo, Huang and Metaxas [68], and Jain, Ye and Liu [54].

10.2.4 Muscles

As with PD actuators, specifying torques directly is not particularly realistic from a biomechanical point of view. Humans and other animals possess the sense of proprioception, the sense of knowing where different parts of the body are and what they are doing. Changes in the length of muscle fibres and their rate of change are detected by muscle spindles, and tension generated in the muscle is detected by Golgi tendon organs (see for example Nigg and Herzog [81]). With all this information available it seems probable that it is used in some way in low-level control of movement, and a prominent idea along these lines is the equilibrium point hypothesis. This has been referred to in the context of motion synthesis by Ngo and Marks [79] and Neff and Fiume [78].

Introduced in 1966 by Feldman [28], the equilibrium point hypothesis posits that the brain positions a joint by specifying threshold lengths beyond which the muscles will produce force, and thus the equilibrium point of a joint (the point to which the limb will settle if the force-length properties of its muscles remain fixed). Movement can then be controlled by supplying a "virtual trajectory", a smoothly varying kinematic path, for the equilibrium point. See for example Shadmehr [104] for a brief overview and Latash [63] for (much) more detail.

The equilibrium point hypothesis concerns the signals sent by the brain to control movement, and regardless of whether it turns out to be true the ac-

tual force production occurs in muscles. Unlike the actuators described above, muscles can only pull, not push, and while the biological situation can be approximated by pairs of opposed muscles, reality is (as usual) more complicated. In general multiple muscles cross most joints and some muscles cross multiple joints, providing different force and energy characteristics and different lines of action.

Broadly speaking, a muscle-tendon unit consists of a contractile muscle body in series with an elastic tendon. The force exerted by a muscle depends on activation level (which has dynamics of its own), current length and current shortening velocity nonlinearly, see e.g. [81] (and non-independently [48], though this is almost always ignored). Muscle and tendon dynamics are often simulated using the flexible and much-cited dimensionless model of Zajac [139], but finding appropriate formulas for the non-linear relationships and parameter values for everything can be difficult, so we refer to a selection of relevant papers here. Thelen [113] presents simplified formulas with specific coefficients, while Brown, Cheng and Loeb [14] present more complex formulas with best fit partition based on biological (feline) measurements. Anderson and Pandy [4] and Garner and Pandy, 2001 [37] present muscle length and force values for a variety of muscles in the human leg and arm respectively. Garner and Pandy, 2003 [38] present a method for estimating muscle parameters and compare their human upper limb results to anatomically measured results by a number of other research groups. A large collation of morphological data may be found in [137], and more recently authors such as Garner and Pandy [35] have published detailed results based on the US National Library of Medicine's Visible Human Project.

The muscle-tendon unit connects one or more *origin* regions on bones to one or more *insertion* regions on other bones, sliding around bones, other muscles, and other tissue influencing its path. In many mathematical models, origin and insertion regions are treated as points and muscles with multiple origins or insertions, or whose origin or insertion regions are large, are often broken into multiple separate muscles with single origin and insertion points. In these models, muscles are treated as lines rather than volumetrically, and are given simple objects such as cylinders and spheres to slide around, and *via* points fixed relative to a bone through which the muscle line must pass. See Garner and Pandy, 2000 [36] for details.

This all adds a great deal of complexity and computational expense, and much more detail can be added still. Is it worthwhile?

One advantage of this is that it is more realistic in terms of the ways the properties of muscle and tendon affect movement. In [64, page 196], Latash writes that "as early as 1938, Sir A.V. Hill demonstrated that the mass-spring model misrepresented basic dynamic properties of muscles", and Buehrmann

and Di Paolo even made the main title of their 2006 paper [15] *Biological actuators are not just springs*. Pandy writes in [90] (from a biomechanics point of view) that "[t]here is some evidence to suggest that musculoskeletal geometry (i.e. muscle paths) is the most critical element of the modeling process". McNeill Alexander begins his synopsis of [2] with "Large mammals save much of the energy they would otherwise need for running by means of elastic structures in their legs." Muscle and tendon are found to be the elastic structures storing most energy, and their geometry is of considerable importance.

Another advantage is that it may be easier to control. Buehrmann and Di Paolo [15] "find that the various non-linearities of the model lead to desirable properties with regard to controllability, such as increased stability and robustness to noise, independence of position and stiffness, or near linearity in search space."

In the field of biomechanics, realistic models of parts of the human body have been constructed with this level of detail, and appropriate control signals have been found at great computational expense. Anderson and Pandy [4] survey a number of earlier works, and their own jumping control optimisation used a supercomputer and involved calculations totalling multiple CPU-months, mostly for numerical estimation of derivatives.

Musculoskeletal models have also seen some use in motion synthesis, mostly focusing on one area of the body, for example Komura, Shinagawa and Kunii [59, 60], legs, or Lee and Terzopoulos [66], the neck. Weinstein [126] and Weinstein, Guendelman and Fedkiw [125] actuate the entire body using PD controllers with muscle geometry.

All the complexities of physical modelling add up to a lot of computation time, especially since many thousands of simulations may need to be run. If some or most of these simulations could be replaced by cheaper approximations, then the whole process would proceed much more quickly. This is our motivation. A general approximation of articulated body physics could dramatically speed up many different motion synthesis approaches.

Chapter 11

Approximation

Many approaches to control involve an implicit or explicit approximation to the system's dynamics. In the field of motion synthesis, implicit approximation has been used in neural networks by, for example, Smith [107], but such an implicit approximation is inaccessible as it is entwined with the control logic for moving. The only major work in motion synthesis on explicitly approximating model dynamics has been by Radek Grzeszczuk and co-authors, who used neural networks. This work was preceded by functionally similar research in robotics such as that of Andrew Moore and co-authors, who used memory-based local methods.

In Grzeszczuk's 1998 PhD thesis [41] and the related paper of the same year by Grzeszczuk, Terzopoulos and Hinton [43], a two stage approach is described. Firstly, a neural network is trained to predict the model state after a fixed time step when given a starting state, control signal and external forces. The training data comes from a physics simulator running with "typical control input" which comes from a separate motion synthesis process related to Grzeszczuk and Terzopoulos [42], and the system is restricted to continuous functions. Secondly, the physics simulator is set aside and controllers are optimised for using only the approximation. The resulting controllers can be applied in either the physics simulator or its approximation, "yielding animations that in most cases differ only minimally". The authors note that the derivatives available from the approximation make much more efficient optimisation possible, producing speed-ups of more than two orders of magnitude.

In Moore's 1990 PhD dissertation [72] and a series of related papers, memory-based methods are described in which experiences are explicitly stored as (*state*, *action*, *behaviour*) triples. Simple local approximations are constructed on evaluation, the value at the nearest known point in [72] and Moore 1991 [73], nearest known point, locally weighted averages and locally weighted regression in

Moore 1992 [74], and locally weighted polynomials or locally weighted regression in Moore, Atkeson, and Schaal 1995 [75] and Atkeson, Moore, and Schaal 1997 [6, 5]. The function which is approximated can be the forward model as in Grzeszczuk's work, (*state*, *action*) $\mapsto$ *behaviour*, the inverse model, (*state*, *behaviour*) $\mapsto$ *action*, or both. Using the inverse model can be faster but may also be problematic as it may not be a function, and in [74] it is shown that learning only the forward model can be sufficient.

Reitsma and Pollard [94] showed, in the context of motion capture, that people are sensitive to errors in animated human motion, and this motivates our concern for the accuracy of our approximation. Grzeszczuk et al and Moore et al, respectively, have dealt primarily and entirely with non-human motion.

This approximation approach is somewhat analogous to surrogate optimisation, except that here the entire behaviour of the system is approximated, rather than just the score being optimised. Here, the score function changes depending on the motion being constructed, and so a system behaviour approximation can be reused for different motions.

In this project we investigate the possibility of approximating the forward model with Radial Basis Functions. As with Grzeszczuk et al and Moore et al we have no intention of trying to learn every possible behaviour of the system, but only the parts relevant to the motion we are interested in producing. Our basic approach is shown in Algorithm 8.

Algorithm 8 Approximation Cache

Input : Starting state $\boldsymbol{s}$, control signal $\boldsymbol{u}$
Output: Approximation of behaviour resulting from $(\boldsymbol{s}, \boldsymbol{u})$ which is close to the true behaviour $\boldsymbol{b}$

1: **if** our approximation exists at $(\boldsymbol{s}, \boldsymbol{u})$ and is predicted to be accurate there **then**
2: **return** the approximate behaviour at $(\boldsymbol{s}, \boldsymbol{u})$, $\boldsymbol{b}^*$.
3: **else**
4: Calculate the true behaviour at $(\boldsymbol{s}, \boldsymbol{u})$, $\boldsymbol{b}$.
5: Add $(\boldsymbol{s}, \boldsymbol{u}) \mapsto \boldsymbol{b}$ to our approximation.
6: **return** $\boldsymbol{b}$.
7: **end if**

Our approximation is built up gradually, around the areas the optimiser explores. The method relies on the ability to add new data one or a few observations at a time, and an estimate of how accurate the approximation is, i.e. when it is safe to depend on it. We were ultimately unsuccessful in finding a viable accuracy estimation method, so several of the other ideas presented here are in early stages and contain known flaws, there being no reason yet to fix them.

11.1 Abstraction

We shift and scale the state and control variables so that they each lie in the range $[0, 1]$. Then we treat the physics simulator as an unknown function f which takes as input a vector $\boldsymbol{x} \in [0, 1]^d$ consisting of the current model state, the control parameters, and possibly the length of the simulated motion snippet, which produces a new model state $\boldsymbol{y} \in [0, 1]^{d_s}$ as output.

$$f \colon [0, 1]^d \mapsto [0, 1]^{d_s}.$$

The actions of external forces such as gravity are implicitly specified by the model's state. While in general this function will be discontinuous, particularly as a result of collisions, for this project we take care to have only soft collisions at joint limits, and no collisions with other objects. Some possibilities for discontinuous motion remain, for example a pendulum swinging up to a vertical position could fall in either direction with a little more or less velocity, and we avoid situations of this type with well-chosen joint limits. We therefore have our f being a function with high dimensional input and output, continuous on our domain of interest. This is suitable for approximation with RBFs.

11.2 Approximation

We approximate the simulator with standard pointwise RBF methods, with d_s independent 1-dimensional interpolants (one for each output dimension) of the form

$$s(\boldsymbol{x}) = \sum_{i=1}^{n} c_i \Phi(\boldsymbol{x} - \boldsymbol{x}_i) + p(\boldsymbol{x}),$$

where Φ is radial ($\Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|_2)$, where $\phi \colon \mathbb{R}^+ \mapsto \mathbb{R}$) and strictly conditionally definite of order k, $p \in \pi^d_{k-1}$, the space of d-variate polynomials with total degree at most $k - 1$, meaning that

$$\sum_{i,j=1}^{n} c_i c_j \phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) > 0,$$

for all $\boldsymbol{c} \neq \boldsymbol{0}$ such that

$$\sum_{i=1}^{n} c_i q(\boldsymbol{x}_i) = 0, \quad \forall\, q \in \pi^d_{k-1}.$$

We begin with the pointwise analogs of Problem 2, page 10, and parts of Section 3.3, page 17. Let ℓ be the dimension of π^d_{k-1}, and let $\{p_j | j = 1, \ldots, \ell\}$

be a basis for it which is biorthogonal to point evaluations at $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ (i.e. $p_j(\boldsymbol{x}_i) = \delta_{ij}$ for $i, j \in \{1, \ldots, \ell\}$). Then, setting

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{i=1}^{\ell} a_i p_i(\boldsymbol{x}) \\ P_{n\times\ell} &= [P_{ij}] = [p_j(\boldsymbol{x}_i)] \\ G_{n\times n} &= [G_{ij}] = [\phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2)]\,, \end{aligned}$$

we can encapsulate all the above requirements in the following equation.

$$\begin{bmatrix} G & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \tag{11.1}$$

Following [10], as in the integral interpolation case on page 18, we transform (11.1) into a positive definite system

$$Q^T G Q \boldsymbol{\gamma} = Q^T \boldsymbol{b} \tag{11.2}$$

where

$$Q = \begin{bmatrix} \hat{Q} \\ I_{n-\ell} \end{bmatrix}, \text{ for } \hat{Q}_{\ell\times(n-\ell)} = \left[\hat{Q}_{ij}\right] = [-p_i(x_{\ell+j})]\,.$$

We solve the interpolation problem using Algorithm 9.

Algorithm 9 The Interpolation Process

1: Find $\boldsymbol{\gamma}$ by solving the positive definite system (11.2).
2: Set $\boldsymbol{c} = Q\boldsymbol{\gamma}$.
3: Find p by interpolating to the residual at $\{\boldsymbol{x}_i | i = 1, \ldots, \ell\}$.

11.3 Matrix Updating

Unlike in the track data application of Section 5.1, page 43, here we are only adding points, not removing them, so we can use matrix updating to make improving the approximation efficient. The matrix Q is naturally partitioned,

$$Q = \left[\frac{\hat{Q}}{I_{n-\ell}}\right],$$

and we can take advantage of this structure to speed up the matrix multiplications needed for Equation (11.2). Let us divide G and $\boldsymbol{b}$ up similarly:

$$G = \left[\begin{array}{c|c} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right], \boldsymbol{b} = \left[\begin{array}{c} b_1 \\ \hline b_2 \end{array}\right],$$

where G_{11} is $\ell \times \ell$, G_{12} is $\ell \times (n-\ell)$, G_{21} is $(n-\ell) \times \ell$, G_{22} is $(n-\ell) \times (n-\ell)$, b_1 is $\ell \times 1$, and b_2 is $(n-\ell) \times 1$. Then we can multiply out the blocks as follows to get more efficient formulas for both sides of Equation (11.2):

$$\begin{aligned} Q^T G Q &= \left[\begin{array}{cc} \hat{Q}^T & I_{n-\ell} \end{array}\right] \left(\left[\begin{array}{cc} G_{11} & G_{12} \\ G_{21} & G_{22} \end{array}\right] \left[\begin{array}{c} \hat{Q} \\ I_{n-\ell} \end{array}\right] \right) \\ &= \left[\begin{array}{cc} \hat{Q}^T & I_{n-\ell} \end{array}\right] \left(\left[\begin{array}{c} G_{11}\hat{Q} + G_{12} \\ G_{21}\hat{Q} + G_{22} \end{array}\right] \right) \\ &= \hat{Q}^T G_{11} \hat{Q} + \hat{Q}^T G_{12} + G_{21}\hat{Q} + G_{22} \end{aligned} \tag{11.3}$$

and

$$\begin{aligned} Q^T \boldsymbol{b} &= \left[\begin{array}{cc} \hat{Q}^T & I_{n-\ell} \end{array}\right] \left[\begin{array}{c} b_1 \\ b_2 \end{array}\right] \\ &= \hat{Q}^T b_1 + b_2 \end{aligned} \tag{11.4}$$

We can use this same matrix decomposition to do updating. If we want to add m new points $x_{n+1} \ldots x_{n+m}$ to the system, we extend Q, G and $\boldsymbol{b}$ like so:

$$Q' = \left[\begin{array}{c|c} \hat{Q} & q \\ \hline I_{n-\ell} & 0 \\ \hline 0 & I_m \end{array}\right], G' = \left[\begin{array}{c|c|c} G_{11} & G_{12} & g_{13} \\ \hline G_{21} & G_{22} & g_{23} \\ \hline g_{31} & g_{32} & g_{33} \end{array}\right], \boldsymbol{b}' = \left[\begin{array}{c} b_1 \\ \hline b_2 \\ \hline b_3 \end{array}\right],$$

where q is $\ell \times m$, g_{*3} are m elements wide, and b_3 and g_{3*} are m elements high.

Our updated left and right sides of Equation (11.2) are then

$$\begin{aligned} Q'^T G' Q' &= \begin{bmatrix} \hat{Q}^T & I_{n-\ell} & 0 \\ q^T & 0 & I_m \end{bmatrix} \left(\begin{bmatrix} G_{11} & G_{12} & g_{13} \\ G_{21} & G_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} \begin{bmatrix} \hat{Q} & q \\ I_{n-\ell} & 0 \\ 0 & I_m \end{bmatrix} \right) \\ &= \begin{bmatrix} \hat{Q}^T & I_{n-\ell} & 0 \\ q^T & 0 & I_m \end{bmatrix} \left(\begin{bmatrix} G_{11}\hat{Q} + G_{12} & G_{11}q + g_{13} \\ G_{21}\hat{Q} + G_{22} & G_{21}q + g_{23} \\ g_{31}\hat{Q} + g_{32} & g_{31}q + g_{33} \end{bmatrix} \right) \\ &= \begin{bmatrix} Q^T G Q & \hat{Q}^T \left(G_{11}q + g_{13}\right) + G_{21}q + g_{23} \\ q^T \left(G_{11}\hat{Q} + G_{12}\right) + g_{31}\hat{Q} + g_{32} & q^T \left(G_{11}q + g_{13}\right) + g_{31}q + g_{33} \end{bmatrix} \end{aligned} \tag{11.5}$$

and

$$\begin{aligned} Q'^T \boldsymbol{b}' &= \begin{bmatrix} \hat{Q}^T & I_{n-\ell} & 0 \\ q^T & 0 & I_m \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \\ &= \begin{bmatrix} \hat{Q}^T b_1 + b_2 \\ q^T b_1 + b_3 \end{bmatrix} = \begin{bmatrix} Q^T \boldsymbol{b} \\ q^T b_1 + b_3 \end{bmatrix}. \end{aligned} \tag{11.6}$$

Note that we already know $Q^T G Q$ and $Q^T \boldsymbol{b}$ from the previous state, so it is only elements involving the new information which need to be calculated.

We can solve a positive definite system such as (11.2) by Cholesky factorisation, i.e. finding a lower triangular L such that $H = L^T L$ for $H_{n\times n} = [h_{ij}]$ positive definite. $L_{n\times n} = [l_{ij}]$ can be constructed row by row:

$$l_{11} = \sqrt{h_{11}} \tag{11.7}$$

$$l_{ij} = \left(h_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj} \quad \text{for } i > 1, j < i \tag{11.8}$$

$$l_{ii} = \sqrt{h_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \qquad \text{for } i > 1. \tag{11.9}$$

Now when m rows and columns are added to H, the Cholesky factor L can be updated with m new rows, one by one, using equations (11.8) and (11.9).

11.4 Scaling

Adding points with matrix updating is much more efficient than fitting the entire approximation every time, but each additional point increases the costs

of updating, storage and evaluation. A common solution, as used in Part II, is to break things up into multiple smaller approximations, and blend them together to cover the domain of interest. In this application, however, we have an unusual advantage in that we do not actually need continuity, letting us leave out the most complicated part of piecewise methods, the overlapping subdomains. Optimisers move in discrete jumps, and if the accuracy estimator is working, approximations *used* on either side of a boundary between adjacent subdomains will agree very closely. Admittedly, the boundary discontinuities will create a few shallow local minima, which we may need to be wary of.

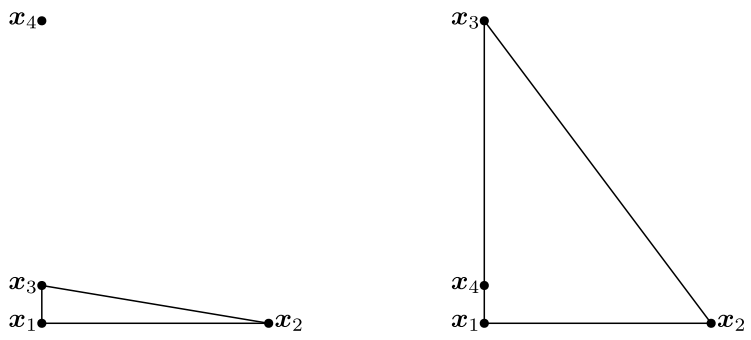
With this knowledge in hand, we take the simplest approach possible in our multidimensional setting and use a kd-tree, see e.g. Samet [99]. When the number of points in a tree cell reaches a pre-defined limit (e.g. 200), we call that cell full and subdivide it. "On the axis the points are most spread out on" and "at the median point" are the rules most often followed. We have tried others such as "as close to the centre as possible while respecting a minimum number of points per subdomain", which keeps cells more cubical, but we do not have enough evidence to recommend one particular strategy for this application.

Whichever division scheme is chosen, we have a choice as to whether the approximations in the two new subdomains *replace* the approximation in the newly subdivided cell, or whether they *refine* it. The former has the advantage in terms of memory usage and evaluation speed and seemed a clear winner, until with the hindsight of having completed the research for the multilevel surface reconstruction in Part II, we realised that the other approach might help with our accuracy estimation difficulties (see Section 11.6).

Whether subdomain approximations replace or refine, they need to start afresh, and there are two important problems which can occur, and which are more likely in this application than in other areas such as surface reconstruction. Firstly, the points in a new subcell, even if there are enough of them, may all lie on a lower dimensional subspace. Secondly, as well as subsets of points being on subspaces, some points may be very close together, meaning the points need to be re-ordered for creating a polynomial basis for π^d_{k-1}, biorthogonal to point evaluations at $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$, to be possible or to not lead to poor conditioning.

Some optimisation techniques explore multidimensional space along straight lines, creating regions where the differences between points do not span the full space. If all the points in a tree cell are on such a subspace of $[0,1]^d$ then creating a d-dimensional RBF is impossible (as we cannot satisfy the unisolvency for π^d_{k-1} requirement, see Definition 3, page 9) and we have a choice between approximating only on the subspace, waiting for points off the subspace to be added, or automatically adding some artificial points.

Approximating only on the subspace means we would need to check each



(a) Bad ordering.

(b) Good ordering.

Figure 11.1: Bad and good choices for polynomial basis special points.

added point and restart the approximation if a point is off the subspace (the polynomial basis will change, which in our updating scheme changes everything). If we wait for points not on the subspace to appear, we may end up never constructing an approximation, potentially wasting resources as the physics simulator is used instead. Alternatively, we can use a Gram-Schmidt process to find a basis for $[0,1]^d$ which includes a basis for our problematic subspace, and add in new points some distance away from the subspace.

As for the second problem, consider the case of $\mathbb{R}^2$ and linear polynomials. Here, $\ell = 3$, so we choose 3 points to be special, $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, and find linear basis functions $p_1(\boldsymbol{x}), p_2(\boldsymbol{x}), p_3(\boldsymbol{x})$ such that $p_j(\boldsymbol{x}_i) = \delta_{ij}$. In this case, each p_i will simply be a ramp going from 1 at $\boldsymbol{x}_i$ down to 0 on the side opposite $\boldsymbol{x}_i$ of the triangle formed by $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ (see Figure 11.1). To calculate entries of $\hat{Q}$ we need to evaluate the polynomial basis at the non-special points (just $\boldsymbol{x}_4$ in the figure). When the points are ordered as in Figure 11.1a, the value of $p_1(\boldsymbol{x}_4)$ will be large, whereas with the Figure 11.1b ordering, it will be between 0 and 1. In practice the distance and value differences may be much more extreme than can be shown in a figure, and a good choice of ordering can make a significant difference in the conditioning of the system in Equation (11.2).

A good heuristic for automatically choosing the polynomial basis special points is to try to maximise the area of the triangle. This is equivalent to maximising the area (volume) of the parallelogram (parallelepiped) defined by $\boldsymbol{x}_1$ and the vectors from $\boldsymbol{x}_1$ to the other special points, which is given by the

absolute value of the determinant

$$\left| \begin{array}{ccc} (\boldsymbol{x}_2 - \boldsymbol{x}_1) & \cdots & (\boldsymbol{x}_\ell - \boldsymbol{x}_1) \end{array} \right|.$$

This is quite cheap to calculate since ℓ is always small, so a simple strategy is just to try a few permutations of (all) the points, and keep the ordering for which the first ℓ produce the largest volume. The number of trials needed will vary depending on the nature of the data.

This volume maximising strategy can also be viewed as minimising the norm of the polynomial interpolation operator. Let D be our domain, a compact set. Let $\mathcal{L}\colon C(D) \mapsto S$ be an operator which approximates $g \in C(D)$ by $\sigma \in S$, where S is finite dimensional (for our purposes here, $S = \pi_1^d$). Then the interpolant to g at $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ is

$$(\mathcal{L}g)(\boldsymbol{x}) = \sum_{i=1}^{\ell} g(\boldsymbol{x}_i) p_i(\boldsymbol{x}),$$

and

$$|(\mathcal{L}g)(\boldsymbol{x})| \leq \sum_{i=1}^{\ell} |g(\boldsymbol{x}_i)| \, |p_i(\boldsymbol{x})| \,. \tag{11.10}$$

The ∞-norm of $\mathcal{L}$ is defined as

$$\|\mathcal{L}\|_\infty = \sup_{g:\|g\|_\infty = 1} \|\mathcal{L}g\|_\infty, \tag{11.11}$$

and since we are on a compact domain,

$$\|\mathcal{L}\|_\infty = \max_{g:\|g\|_\infty = 1} \|\mathcal{L}g\|_\infty \tag{11.12}$$

$$\leq \max_{x \in D} \sum_{i=1}^{\ell} |p_i(\boldsymbol{x})| \,, \tag{11.13}$$

from (11.10). Clearly, we cannot let the polynomial basis get too large on the domain D. (That this is a lower bound as well as an upper bound can be shown by constructing a suitable continuous g with $\|g\|_\infty = 1$.)

The approximations we use here interpolate to known values since as the results of deterministic simulations they really are the true values of f. In between known points, however, it is a different story, and it would be nice to know where our approximation is accurate enough to be relied upon, and where it is untrustworthy and needs to have new true points calculated and added.

We investigate this issue in the next section.

11.5 Approximation Accuracy

Grzeszczuk et al and Moore et al take very different approaches to the accuracy of their approximations. Grzeszczuk's approximations are computed offline ahead of time, and rely on model-specific neural network structure, good sampling and an error reduction technique for flexible bodied models to achieve accuracy over the domain of interest. Errors are not explicitly calculated or estimated during the training phase.

In [73], Moore describes an explicit estimate for the probability that the nearest neighbour approximation will be accurate enough, and the use of such estimates in the automatic exploration of the system space en route to finding an appropriate control signal. The estimate is based on the distance to the nearest known point. Schaal and Atkeson, 1994 [100] take this idea further, exploiting the existing body of statistical analysis available for locally weighted regression.

We experimented with a variety of distance-based trust heuristics for RBF approximations to both synthetic and articulated body physics simulation data, and while we achieved moderate success in one dimension, and some hard-won success in two dimensions, our approach did not usefully generalise further. This led us to reconsider our core assumption, that error would be closely related to distance from known data points. At the data points, the approximation interpolates, so we have at least some small neighbourhood of very low error around each point, but our numerical experiments have shown that the rate at which error grows with distance from a data point varies so much that the distance is much of the time simply not useful for predicting error.

Figure 11.2 shows the results of one experiment, though different models, optimisers, sampling methods, and basic functions produce comparable results. Here, the model we approximate is a double pendulum with a PD actuator at each hinge joint and under the influence of gravity ($-9.81\,\mathrm{ms}^{-2}$), and its goal is to move from near hanging straight down to having both joints bent near 60° from straight after 0.25 seconds. The two parts of the pendulum are cylinders 40 cm long and 3 cm in radius, have a density of 0.8 kg/l, are connected with parallel hinges end-to-end and to the environment, and are unable to collide with each other. The actuators have $k_p = 200\,\mathrm{Nm/rad}$, $k_d = 10\,\mathrm{Nms/rad}$, and maximum torque 20 Nm. The model's state is 4-dimensional (two angles and their derivatives) and the control signal is 2-dimensional, so $f\colon [0,1]^6 \mapsto [0,1]^4$, and 2-dimensional optimisation is needed.

To obtain some points with which to approximate, we perform 10 optimi-

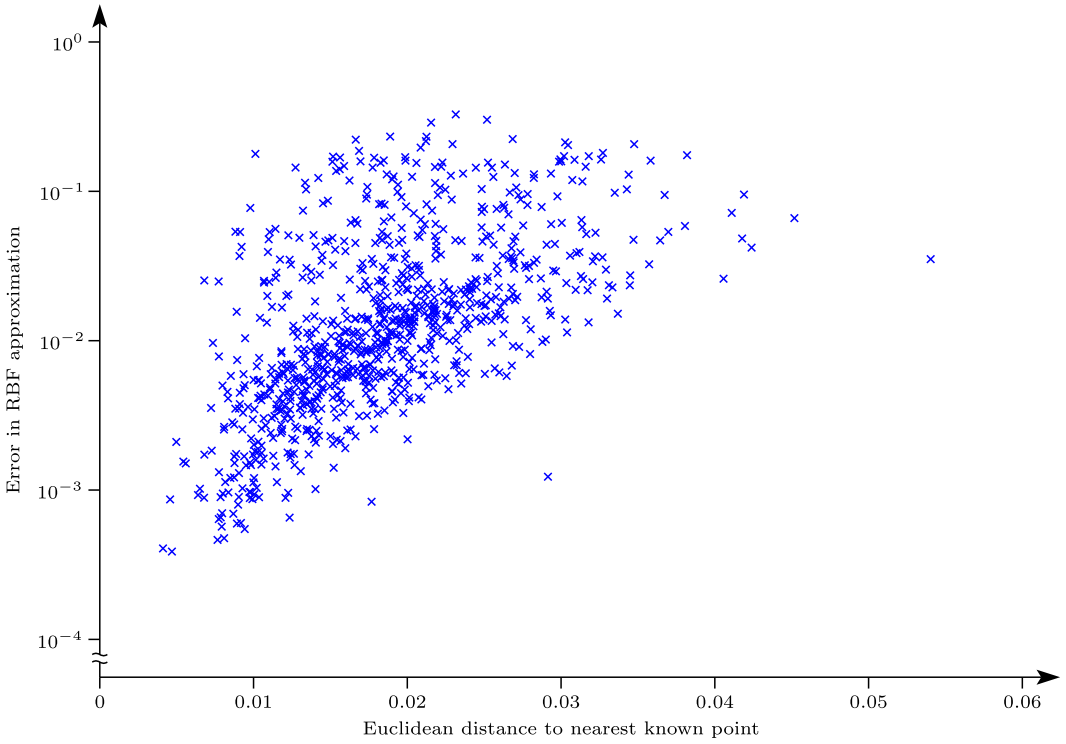


Figure 11.2: Error versus nearest point distance in the $[0,1]^6 \mapsto [0,1]^4$ actuated double pendulum experiment.

sation runs using the downhill simplex method, perturbing the start state and optimisation start point by normally distributed random numbers with mean 0 and standard deviation 0.01. We take the points which differ by more than 10^{-6} along any axis (roughly $1,000$) and construct an RBF interpolant using multiquadrics with parameter 10^{-6} and a linear polynomial term. We perturb the approximation centres by a normally distributed amount (again, mean 0 and standard deviation 0.01) to generate the same number of evaluation points, and in the figure we plot the Euclidean norm of the approximation error at these points versus the Euclidean distance to their nearest approximation centre.

It can be seen from the figure that distance from known points is not a good predictor of approximation error.

11.6 Conclusions and Future Directions

It remains unclear whether or not this kind of RBF approach to speeding up motion synthesis is viable. In principle it should be possible to do at least as well as the neural networks of Grzeszczuk and colleagues, but this may only be feasible if a similar strategy is used, trusting accuracy to secondary processes rather than estimating error directly, relying on well distributed training data rather than learning as an optimiser progresses, and building structure into the approximation which matches the model being simulated.

An important future direction is to explore the possibilities of accuracy estimation based on a multilevel approximation. The basic idea of this is to use the sizes of the approximate residuals, calculated as one descends the levels during an evaluation, as an estimate of the error. If one reaches the bottom (or a higher level which is still allowed more points) and has an estimated error greater than some ϵ, then the simulator is called on to provide a new data point. Preliminary experiments with the $[0,1]^6 \mapsto [0,1]^4$ actuated double pendulum system suggest that this approach may be more fruitful.

Another limitation of our approach is its restriction to continuous mappings. A lot of real, desirable movement depends crucially on discontinuous or near-discontinuous action—the running and ball-hitting of a tennis player, for example. One possible approach would be to map a collision or discontinuity as a kind of surface in (*state*, *action*) space, detect when this surface is crossed, and switch to another approximation. In this way, discontinuities would be explicitly represented.

As far as motion synthesis in general goes, while there are many more things achievable now than there were two decades ago when the field was brand new, a lot of work remains to be done, and approximation almost certainly has a part to play.

Appendix A

Ball Source Formulas

In this appendix we present a selection of ball source functions derived using the techniques described in Section 4.2. These formulas can be used to interpolate to ball integral data in $\mathbb{R}^3$ and $\mathbb{R}^5$. There is some minor variation in formatting as the formulas are generated and formatted automatically, so as to avoid the possibility of transcription errors.

Gaussian ball source

$$\Phi(\boldsymbol{x}) = e^{-\nu^2\|\boldsymbol{x}\|^2}, \quad \boldsymbol{x} \in \mathbb{R}^3.$$

$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \frac{3}{8c^3\nu^4\|\boldsymbol{x}\|}\Big(e^{-\nu^2(\|\boldsymbol{x}\|+c)^2} - e^{-\nu^2(\|\boldsymbol{x}\|-c)^2} + \sqrt{\pi}\,\nu\|\boldsymbol{x}\|\Big(\operatorname{erf}\big(\nu(\|\boldsymbol{x}\|+c)\big) - \operatorname{erf}\big(\nu(\|\boldsymbol{x}\|-c)\big)\Big)\Big).$$

$$\Phi(\boldsymbol{x}) = e^{-\nu^2\|\boldsymbol{x}\|^2}, \quad \boldsymbol{x} \in \mathbb{R}^5.$$

$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \frac{15}{32c^5\nu^8\|\boldsymbol{x}\|^3}\Big(e^{-\nu^2(\|\boldsymbol{x}\|+c)^2}\left(-1-2c\|\boldsymbol{x}\|\nu^2+2\|\boldsymbol{x}\|^2\nu^2\right) - e^{-\nu^2(\|\boldsymbol{x}\|-c)^2}\left(-1+2c\|\boldsymbol{x}\|\nu^2+2\|\boldsymbol{x}\|^2\nu^2\right) + 2\sqrt{\pi}\nu^3\|\boldsymbol{x}\|^3\Big(\operatorname{erf}\big(\nu(\|\boldsymbol{x}\|+c)\big) - \operatorname{erf}\big(\nu(\|\boldsymbol{x}\|-c)\big)\Big)\Big).$$

$(\Phi \star \mathcal{B}_c \star \mathcal{B}_c)(\boldsymbol{x})$ and $(\Phi \star \mathcal{B}_c \star \mathcal{B}_d)(\boldsymbol{x})$ can be found analytically, but unfortunately the formulas are very long and complex.

Linear/biharmonic ball source

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|, \quad \boldsymbol{x} \in \mathbb{R}^3.$$

$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \begin{cases} \frac{3c}{4} + \frac{\|\boldsymbol{x}\|^2}{2c} - \frac{\|\boldsymbol{x}\|^4}{20c^3}, & 0 \le \|\boldsymbol{x}\| < c, \\ \frac{c^2}{5\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_c)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{36c}{35} + \frac{2\|\boldsymbol{x}\|^2}{5c} - \frac{\|\boldsymbol{x}\|^4}{20c^3} + \frac{\|\boldsymbol{x}\|^5}{80c^4} - \frac{\|\boldsymbol{x}\|^7}{4480c^6}, & 0 \le \|\boldsymbol{x}\| < 2c, \\ \frac{2c^2}{5\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & 2c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_d)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{-3d^4}{140c^3} + \frac{3d^2}{10c} + \frac{3c}{4} - \frac{d^2\|\boldsymbol{x}\|^2}{10c^3} + \frac{\|\boldsymbol{x}\|^2}{2c} - \frac{\|\boldsymbol{x}\|^4}{20c^3}, & 0 \le \|\boldsymbol{x}\| < c-d, \\ B_{-1}^{cd}\|\boldsymbol{x}\|^{-1} + B_0^{cd} + B_1^{cd}\|\boldsymbol{x}\| + B_2^{cd}\|\boldsymbol{x}\|^2 + B_3^{cd}\|\boldsymbol{x}\|^3 & \\ \quad + B_4^{cd}\|\boldsymbol{x}\|^4 + B_5^{cd}\|\boldsymbol{x}\|^5 + B_7^{cd}\|\boldsymbol{x}\|^7, & c-d \le \|\boldsymbol{x}\| < d+c, \\ \frac{d^2}{5\|\boldsymbol{x}\|} + \frac{c^2}{5\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & d+c \le \|\boldsymbol{x}\|, \end{cases}$$

where

$$\begin{aligned} B_{-1}^{cd} &= \tfrac{(d-c)^6\left(d^2+6cd+c^2\right)}{640c^3d^3}, \\ B_0^{cd} &= \tfrac{-3(d+c)^5\left(d^2-5cd+c^2\right)}{280c^3d^3}, \\ B_1^{cd} &= \tfrac{(d-c)^4\left(d^2+4cd+c^2\right)}{32c^3d^3}, \\ B_2^{cd} &= \tfrac{-\left((d+c)^3\right)\left(d^2-3cd+c^2\right)}{20c^3d^3}, \\ B_3^{cd} &= \tfrac{3(d-c)^2(d+c)^2}{64c^3d^3}, \\ B_4^{cd} &= \tfrac{-(d+c)\left(d^2-cd+c^2\right)}{40c^3d^3}, \\ B_5^{cd} &= \tfrac{d^2+c^2}{160c^3d^3}, \\ B_7^{cd} &= \tfrac{-1}{4480c^3d^3}. \end{aligned}$$

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|, \quad \boldsymbol{x} \in \mathbb{R}^5.$$

$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \begin{cases} \frac{5c}{6} + \frac{\|\boldsymbol{x}\|^2}{2c} - \frac{\|\boldsymbol{x}\|^4}{14c^3} + \frac{\|\boldsymbol{x}\|^6}{126c^5}, & 0 \le \|\boldsymbol{x}\| < c, \\ \frac{-c^4}{63\|\boldsymbol{x}\|^3} + \frac{2c^2}{7\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_c)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{800c}{693} + \frac{8\|\boldsymbol{x}\|^2}{21c} - \frac{2\|\boldsymbol{x}\|^4}{49c^3} + \frac{\|\boldsymbol{x}\|^6}{126c^5} - \frac{\|\boldsymbol{x}\|^7}{448c^6} + \frac{\|\boldsymbol{x}\|^9}{16128c^8} & \\ \quad - \frac{\|\boldsymbol{x}\|^{11}}{827904c^{10}}, & 0 \le \|\boldsymbol{x}\| < 2c, \\ \frac{-32c^4}{441\|\boldsymbol{x}\|^3} + \frac{4c^2}{7\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & 2c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_d)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{5d^6}{1386c^5} - \frac{5d^4}{126c^3} + \frac{5d^2}{14c} + \frac{5c}{6} + \frac{d^4\|\boldsymbol{x}\|^2}{42c^5} - \frac{d^2\|\boldsymbol{x}\|^2}{7c^3} + \frac{\|\boldsymbol{x}\|^2}{2c} & \\ \quad + \frac{3d^2\|\boldsymbol{x}\|^4}{98c^5} - \frac{\|\boldsymbol{x}\|^4}{14c^3} + \frac{\|\boldsymbol{x}\|^6}{126c^5}, & 0 \le \|\boldsymbol{x}\| < c-d, \\ B_{-3}^{cd}\|\boldsymbol{x}\|^{-3} + B_{-1}^{cd}\|\boldsymbol{x}\|^{-1} + B_0^{cd} + B_1^{cd}\|\boldsymbol{x}\| + B_2^{cd}\|\boldsymbol{x}\|^2 & \\ \quad + B_3^{cd}\|\boldsymbol{x}\|^3 + B_4^{cd}\|\boldsymbol{x}\|^4 + B_5^{cd}\|\boldsymbol{x}\|^5 + B_6^{cd}\|\boldsymbol{x}\|^6 & c-d \le \|\boldsymbol{x}\| < d+c, \\ \quad + B_7^{cd}\|\boldsymbol{x}\|^7 + B_9^{cd}\|\boldsymbol{x}\|^9 + B_{11}^{cd}\|\boldsymbol{x}\|^{11}, & \\ \frac{-d^4}{63\|\boldsymbol{x}\|^3} - \frac{2c^2d^2}{49\|\boldsymbol{x}\|^3} - \frac{c^4}{63\|\boldsymbol{x}\|^3} + \frac{2d^2}{7\|\boldsymbol{x}\|} + \frac{2c^2}{7\|\boldsymbol{x}\|} + \|\boldsymbol{x}\|, & d+c \le \|\boldsymbol{x}\|, \end{cases}$$

where

$$\begin{aligned}
B_{-3}^{cd} &= \tfrac{(d-c)^{10}\left(d^4+10cd^3+34c^2d^2+10c^3d+c^4\right)}{225792c^5d^5}, \\
B_{-1}^{cd} &= \tfrac{-\left((d-c)^8\right)\left(d^4+8cd^3+22c^2d^2+8c^3d+c^4\right)}{3584c^5d^5}, \\
B_0^{cd} &= \tfrac{5(d+c)^7\left(d^4-7cd^3+17c^2d^2-7c^3d+c^4\right)}{2772c^5d^5}, \\
B_1^{cd} &= \tfrac{-\left((d-c)^6\right)\left(3d^4+18cd^3+38c^2d^2+18c^3d+3c^4\right)}{512c^5d^5}, \\
B_2^{cd} &= \tfrac{(d+c)^5\left(d^4-5cd^3+9c^2d^2-5c^3d+c^4\right)}{84c^5d^5}, \\
B_3^{cd} &= \tfrac{-25(d-c)^4(d+c)^4}{1536c^5d^5}, \\
B_4^{cd} &= \tfrac{(d+c)^3\left(3d^4-9cd^3+11c^2d^2-9c^3d+3c^4\right)}{196c^5d^5}, \\
B_5^{cd} &= \tfrac{-5(d-c)^2(d+c)^2\left(d^2+c^2\right)}{512c^5d^5}, \\
B_6^{cd} &= \tfrac{(d+c)\left(d^4-cd^3+c^2d^2-c^3d+c^4\right)}{252c^5d^5}, \\
B_7^{cd} &= \tfrac{-\left(3d^4+2c^2d^2+3c^4\right)}{3584c^5d^5}, \\
B_9^{cd} &= \tfrac{d^2+c^2}{32256c^5d^5}, \\
B_{11}^{cd} &= \tfrac{-1}{827904c^5d^5}.
\end{aligned}$$

Cubic/triharmonic ball source

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3, \quad \boldsymbol{x} \in \mathbb{R}^3.$$

$$(\Phi \star_3 \mathcal{B}_c)(\boldsymbol{x}) = \begin{cases} \frac{c^3}{2} + \frac{3c\|\boldsymbol{x}\|^2}{2} + \frac{3\|\boldsymbol{x}\|^4}{10c} - \frac{\|\boldsymbol{x}\|^6}{70c^3}, & 0 \le \|\boldsymbol{x}\| < c, \\ \frac{3c^4}{35\|\boldsymbol{x}\|} + \frac{6c^2\|\boldsymbol{x}\|}{5} + \|\boldsymbol{x}\|^3, & c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_c)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{32c^3}{21} + \frac{72c\|\boldsymbol{x}\|^2}{35} + \frac{6\|\boldsymbol{x}\|^4}{25c} - \frac{\|\boldsymbol{x}\|^6}{70c^3} + \frac{3\|\boldsymbol{x}\|^7}{1120c^4} - \frac{\|\boldsymbol{x}\|^9}{33600c^6}, & 0 \le \|\boldsymbol{x}\| < 2c, \\ \frac{72c^4}{175\|\boldsymbol{x}\|} + \frac{12c^2\|\boldsymbol{x}\|}{5} + \|\boldsymbol{x}\|^3, & 2c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_d)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{-d^6}{210c^3} + \frac{9d^4}{70c} + \frac{9cd^2}{10} + \frac{c^3}{2} - \frac{3d^4\|\boldsymbol{x}\|^2}{70c^3} + \frac{3d^2\|\boldsymbol{x}\|^2}{5c} & \\ \quad + \frac{3c\|\boldsymbol{x}\|^2}{2} - \frac{3d^2\|\boldsymbol{x}\|^4}{50c^3} + \frac{3\|\boldsymbol{x}\|^4}{10c} - \frac{\|\boldsymbol{x}\|^6}{70c^3}, & 0 \le \|\boldsymbol{x}\| < c - d, \\ B_{-1}^{cd}\|\boldsymbol{x}\|^{-1} + B_0^{cd} + B_1^{cd}\|\boldsymbol{x}\| + B_2^{cd}\|\boldsymbol{x}\|^2 + B_3^{cd}\|\boldsymbol{x}\|^3 & \\ \quad + B_4^{cd}\|\boldsymbol{x}\|^4 + B_5^{cd}\|\boldsymbol{x}\|^5 + B_6^{cd}\|\boldsymbol{x}\|^6 + B_7^{cd}\|\boldsymbol{x}\|^7 & c - d \le \|\boldsymbol{x}\| < d + c, \\ \quad + B_9^{cd}\|\boldsymbol{x}\|^9, & \\ \frac{3d^4}{35\|\boldsymbol{x}\|} + \frac{6c^2d^2}{25\|\boldsymbol{x}\|} + \frac{3c^4}{35\|\boldsymbol{x}\|} + \frac{6d^2\|\boldsymbol{x}\|}{5} + \frac{6c^2\|\boldsymbol{x}\|}{5} + \|\boldsymbol{x}\|^3, & d + c \le \|\boldsymbol{x}\|, \end{cases}$$

where

$$\begin{aligned} B_{-1}^{cd} &= \tfrac{3(d-c)^8\left(d^2+8cd+c^2\right)}{11200c^3d^3}, \\ B_0^{cd} &= \tfrac{-\left((d+c)^7\right)\left(d^2-7cd+c^2\right)}{420c^3d^3}, \\ B_1^{cd} &= \tfrac{3(d-c)^6\left(d^2+6cd+c^2\right)}{320c^3d^3}, \\ B_2^{cd} &= \tfrac{-3(d+c)^5\left(d^2-5cd+c^2\right)}{140c^3d^3}, \\ B_3^{cd} &= \tfrac{(d-c)^4\left(d^2+4cd+c^2\right)}{32c^3d^3}, \\ B_4^{cd} &= \tfrac{-3(d+c)^3\left(d^2-3cd+c^2\right)}{100c^3d^3}, \\ B_5^{cd} &= \tfrac{3(d-c)^2(d+c)^2}{160c^3d^3}, \\ B_6^{cd} &= \tfrac{-(d+c)\left(d^2-cd+c^2\right)}{140c^3d^3}, \\ B_7^{cd} &= \tfrac{3\left(d^2+c^2\right)}{2240c^3d^3}, \\ B_9^{cd} &= \tfrac{-1}{33600c^3d^3}. \end{aligned}$$

$$\Phi(\boldsymbol{x}) = \|\boldsymbol{x}\|, \quad \boldsymbol{x} \in \mathbb{R}^5.$$

$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \begin{cases} \frac{5c^3}{8} + \frac{3c\|\boldsymbol{x}\|^2}{2} + \frac{9\|\boldsymbol{x}\|^4}{28c} - \frac{\|\boldsymbol{x}\|^6}{42c^3} + \frac{\|\boldsymbol{x}\|^8}{616c^5}, & 0 \le \|\boldsymbol{x}\| < c, \\ \frac{-c^6}{231\|\boldsymbol{x}\|^3} + \frac{c^4}{7\|\boldsymbol{x}\|} + \frac{9c^2\|\boldsymbol{x}\|}{7} + \|\boldsymbol{x}\|^3, & c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_c)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{800c^3}{429} + \frac{160c\|\boldsymbol{x}\|^2}{77} + \frac{12\|\boldsymbol{x}\|^4}{49c} - \frac{2\|\boldsymbol{x}\|^6}{147c^3} + \frac{\|\boldsymbol{x}\|^8}{616c^5} - \frac{\|\boldsymbol{x}\|^9}{2688c^6} \\ \quad + \frac{\|\boldsymbol{x}\|^{11}}{137984c^8} - \frac{3\|\boldsymbol{x}\|^{13}}{28700672c^{10}}, & 0 \le \|\boldsymbol{x}\| < 2c, \\ \frac{-80c^6}{1617\|\boldsymbol{x}\|^3} + \frac{32c^4}{49\|\boldsymbol{x}\|} + \frac{18c^2\|\boldsymbol{x}\|}{7} + \|\boldsymbol{x}\|^3, & 2c \le \|\boldsymbol{x}\|. \end{cases}$$

$$(\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_d)(\boldsymbol{x}) =$$
$$\begin{cases} \frac{5d^8}{8008c^5} - \frac{5d^6}{462c^3} + \frac{5d^4}{28c} + \frac{15cd^2}{14} + \frac{5c^3}{8} + \frac{d^6\|\boldsymbol{x}\|^2}{154c^5} - \frac{d^4\|\boldsymbol{x}\|^2}{14c^3} \\ \quad + \frac{9d^2\|\boldsymbol{x}\|^2}{14c} + \frac{3c\|\boldsymbol{x}\|^2}{2} + \frac{3d^4\|\boldsymbol{x}\|^4}{196c^5} - \frac{9d^2\|\boldsymbol{x}\|^4}{98c^3} + \frac{9\|\boldsymbol{x}\|^4}{28c} & 0 \le \|\boldsymbol{x}\| < c - d, \\ \quad + \frac{d^2\|\boldsymbol{x}\|^6}{98c^5} - \frac{\|\boldsymbol{x}\|^6}{42c^3} + \frac{\|\boldsymbol{x}\|^8}{616c^5}, \\ B_{-3}^{cd}\|\boldsymbol{x}\|^{-3} + B_{-1}^{cd}\|\boldsymbol{x}\|^{-1} + B_0^{cd} + B_1^{cd}\|\boldsymbol{x}\| + B_2^{cd}\|\boldsymbol{x}\|^2 \\ \quad + B_3^{cd}\|\boldsymbol{x}\|^3 + B_4^{cd}\|\boldsymbol{x}\|^4 + B_5^{cd}\|\boldsymbol{x}\|^5 + B_6^{cd}\|\boldsymbol{x}\|^6 \\ \quad + B_7^{cd}\|\boldsymbol{x}\|^7 + B_8^{cd}\|\boldsymbol{x}\|^8 + B_9^{cd}\|\boldsymbol{x}\|^9 + B_{11}^{cd}\|\boldsymbol{x}\|^{11} & c - d \le \|\boldsymbol{x}\| < d + c, \\ \quad + B_{13}^{cd}\|\boldsymbol{x}\|^{13}, \\ \frac{-d^6}{231\|\boldsymbol{x}\|^3} - \frac{c^2d^4}{49\|\boldsymbol{x}\|^3} - \frac{c^4d^2}{49\|\boldsymbol{x}\|^3} - \frac{c^6}{231\|\boldsymbol{x}\|^3} + \frac{d^4}{7\|\boldsymbol{x}\|} + \frac{18c^2d^2}{49\|\boldsymbol{x}\|} \\ \quad + \frac{c^4}{7\|\boldsymbol{x}\|} + \frac{9d^2\|\boldsymbol{x}\|}{7} + \frac{9c^2\|\boldsymbol{x}\|}{7} + \|\boldsymbol{x}\|^3, & d + c \le \|\boldsymbol{x}\|, \end{cases}$$

where

$$\begin{aligned}
B_{-3}^{cd} &= \tfrac{(d-c)^{12}\left(3d^4+36cd^3+146c^2d^2+36c^3d+3c^4\right)}{6623232c^5d^5}, \\
B_{-1}^{cd} &= \tfrac{-\left((d-c)^{10}\right)\left(d^4+10cd^3+34c^2d^2+10c^3d+c^4\right)}{25088c^5d^5}, \\
B_0^{cd} &= \tfrac{5(d+c)^9\left(3d^4-27cd^3+83c^2d^2-27c^3d+3c^4\right)}{48048c^5d^5}, \\
B_1^{cd} &= \tfrac{-9(d-c)^8\left(d^4+8cd^3+22c^2d^2+8c^3d+c^4\right)}{7168c^5d^5}, \\
B_2^{cd} &= \tfrac{(d+c)^7\left(d^4-7cd^3+17c^2d^2-7c^3d+c^4\right)}{308c^5d^5}, \\
B_3^{cd} &= \tfrac{-\left((d-c)^6\right)\left(3d^4+18cd^3+38c^2d^2+18c^3d+3c^4\right)}{512c^5d^5}, \\
B_4^{cd} &= \tfrac{3(d+c)^5\left(d^4-5cd^3+9c^2d^2-5c^3d+c^4\right)}{392c^5d^5}, \\
B_5^{cd} &= \tfrac{-15(d-c)^4(d+c)^4}{2048c^5d^5}, \\
B_6^{cd} &= \tfrac{(d+c)^3\left(3d^4-9cd^3+11c^2d^2-9c^3d+3c^4\right)}{588c^5d^5}, \\
B_7^{cd} &= \tfrac{-9(d-c)^2(d+c)^2\left(d^2+c^2\right)}{3584c^5d^5},
\end{aligned}$$

$$
\begin{aligned}
B_8^{cd} &= \tfrac{(d+c)\left(d^4-cd^3+c^2d^2-c^3d+c^4\right)}{1232c^5d^5},\\
B_9^{cd} &= \tfrac{-\left(3d^4+2c^2d^2+3c^4\right)}{21504c^5d^5},\\
B_{11}^{cd} &= \tfrac{d^2+c^2}{275968c^5d^5},\\
B_{13}^{cd} &= \tfrac{-3}{28700672c^5d^5}.
\end{aligned}
$$

Wendland $\Phi_{3,1}$ ball source

Here, we give formulas only for the $c > s$ case, and leave out the extremely large $\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_d$ formulas.

$$
\begin{aligned}
\Phi(\boldsymbol{x}) &= \left(1-\frac{\|\boldsymbol{x}\|}{s}\right)_+^4 \left(4\frac{\|\boldsymbol{x}\|}{s}+1\right), \quad \boldsymbol{x}\in\mathbb{R}^3.\\
&= \begin{cases} \left(1-\frac{\|\boldsymbol{x}\|}{s}\right)^4\left(4\frac{\|\boldsymbol{x}\|}{s}+1\right), & 0\le\|\boldsymbol{x}\|<s,\\ 0, & s\le\|\boldsymbol{x}\|. \end{cases}
\end{aligned}
$$

$$
\left(\Phi\star_3\mathcal{B}_c\right)(\boldsymbol{x}) =
\begin{cases}
A_0^c + A_2^c\|\boldsymbol{x}\|^2 + A_4^c\|\boldsymbol{x}\|^4 + A_6^c\|\boldsymbol{x}\|^6 + A_8^c\|\boldsymbol{x}\|^8\,, & 0\le\|\boldsymbol{x}\|<c\,,\\
B_{-1}^c\|\boldsymbol{x}\|^{-1} + B_0^c + B_1^c\|\boldsymbol{x}\| + B_2^c\|\boldsymbol{x}\|^2 + B_3^c\|\boldsymbol{x}\|^3 & \\
\quad + B_4^c\|\boldsymbol{x}\|^4 + B_5^c\|\boldsymbol{x}\|^5\,, & c\le\|\boldsymbol{x}\|<s-c\,,\\
C_{-1}^c\|\boldsymbol{x}\|^{-1} + C_0^c + C_1^c\|\boldsymbol{x}\| + C_2^c\|\boldsymbol{x}\|^2 + C_3^c\|\boldsymbol{x}\|^3 & \\
\quad + C_4^c\|\boldsymbol{x}\|^4 + C_5^c\|\boldsymbol{x}\|^5 + C_6^c\|\boldsymbol{x}\|^6 + C_7^c\|\boldsymbol{x}\|^7 & s-c\le\|\boldsymbol{x}\|<s+c\,,\\
\quad + C_8^c\|\boldsymbol{x}\|^8\,, & \\
0\,, & s+c\le\|\boldsymbol{x}\|\,,
\end{cases}
$$

where

$$
\begin{aligned}
A_0^c &= \tfrac{14s^5-84c^2s^3+140c^3s^2-90c^4s+21c^5}{14s^5},\\
A_2^c &= \tfrac{-10(s-c)^3}{s^5},\\
A_4^c &= \tfrac{3(s-c)(2s-3c)}{cs^5},\\
A_6^c &= \tfrac{-2\left(s^2-3c^2\right)}{7c^3s^5},\\
A_8^c &= \tfrac{-1}{42c^3s^5},\\
B_{-1}^c &= \tfrac{4c^4\left(9s^2+c^2\right)}{21s^5},\\
B_0^c &= \tfrac{7s^4-42c^2s^2-45c^4}{7s^4},
\end{aligned}
$$

$$
\begin{aligned}
B_{1}^{c} &= \tfrac{12c^2\left(14s^2+3c^2\right)}{7s^5},\\
B_{2}^{c} &= \tfrac{-10\left(s^2+3c^2\right)}{s^4},\\
B_{3}^{c} &= \tfrac{4\left(5s^2+3c^2\right)}{s^5},\\
B_{4}^{c} &= \tfrac{-15}{s^4},\\
B_{5}^{c} &= \tfrac{4}{s^5},\\
C_{-1}^{c} &= \tfrac{-\left((s+c)^6\right)\left(5s^3-30cs^2+69c^2s-64c^3\right)}{672c^3s^5},\\
C_{0}^{c} &= \tfrac{(s+c)^5\left(s^3-5cs^2+15c^2s-21c^3\right)}{28c^3s^5},\\
C_{1}^{c} &= \tfrac{-3(s+c)^4\left(s^3-4cs^2+17c^2s-48c^3\right)}{56c^3s^5},\\
C_{2}^{c} &= \tfrac{-5(s+c)^3}{s^5},\\
C_{3}^{c} &= \tfrac{(s+c)^2\left(s^3-2cs^2+33c^2s+96c^3\right)}{16c^3s^5},\\
C_{4}^{c} &= \tfrac{-3(s+c)(2s+3c)}{2cs^5},\\
C_{5}^{c} &= \tfrac{-\left(s^3-15c^2s-16c^3\right)}{8c^3s^5},\\
C_{6}^{c} &= \tfrac{s^2-3c^2}{7c^3s^5},\\
C_{7}^{c} &= \tfrac{-15}{224c^3s^4},\\
C_{8}^{c} &= \tfrac{1}{84c^3s^5}.
\end{aligned}
$$

$$
\left(\Phi \star_3 \mathcal{B}_c \star_3 \mathcal{B}_c\right)(\boldsymbol{x}) =
$$

$$
\begin{cases}
A_{0}^{cc} + A_{2}^{cc}\|\boldsymbol{x}\|^2 + A_{4}^{cc}\|\boldsymbol{x}\|^4 + A_{6}^{cc}\|\boldsymbol{x}\|^6 + A_{7}^{cc}\|\boldsymbol{x}\|^7 & \\
\quad + A_{8}^{cc}\|\boldsymbol{x}\|^8 + A_{9}^{cc}\|\boldsymbol{x}\|^9 + A_{11}^{cc}\|\boldsymbol{x}\|^{11}\,, & 0 \le \|\boldsymbol{x}\| < s-2c\,,\\
B_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + B_{0}^{cc} + B_{1}^{cc}\|\boldsymbol{x}\| + B_{2}^{cc}\|\boldsymbol{x}\|^2 + B_{3}^{cc}\|\boldsymbol{x}\|^3 & \\
\quad + B_{4}^{cc}\|\boldsymbol{x}\|^4 + B_{5}^{cc}\|\boldsymbol{x}\|^5 + B_{6}^{cc}\|\boldsymbol{x}\|^6 + B_{7}^{cc}\|\boldsymbol{x}\|^7 & s-2c \le \|\boldsymbol{x}\| < 2c\,,\\
\quad + B_{8}^{cc}\|\boldsymbol{x}\|^8 + B_{9}^{cc}\|\boldsymbol{x}\|^9 + B_{10}^{cc}\|\boldsymbol{x}\|^{10} + B_{11}^{cc}\|\boldsymbol{x}\|^{11}\,, & \\
C_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + C_{0}^{cc} + C_{1}^{cc}\|\boldsymbol{x}\| + C_{2}^{cc}\|\boldsymbol{x}\|^2 + C_{3}^{cc}\|\boldsymbol{x}\|^3 & \\
\quad + C_{4}^{cc}\|\boldsymbol{x}\|^4 + C_{5}^{cc}\|\boldsymbol{x}\|^5 + C_{6}^{cc}\|\boldsymbol{x}\|^6 + C_{7}^{cc}\|\boldsymbol{x}\|^7 & 2c \le \|\boldsymbol{x}\| < s\,,\\
\quad + C_{8}^{cc}\|\boldsymbol{x}\|^8 + C_{9}^{cc}\|\boldsymbol{x}\|^9 + C_{10}^{cc}\|\boldsymbol{x}\|^{10} + C_{11}^{cc}\|\boldsymbol{x}\|^{11}\,, & \\
D_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + D_{0}^{cc} + D_{1}^{cc}\|\boldsymbol{x}\| + D_{2}^{cc}\|\boldsymbol{x}\|^2 + D_{3}^{cc}\|\boldsymbol{x}\|^3 & \\
\quad + D_{4}^{cc}\|\boldsymbol{x}\|^4 + D_{5}^{cc}\|\boldsymbol{x}\|^5 + D_{6}^{cc}\|\boldsymbol{x}\|^6 + D_{7}^{cc}\|\boldsymbol{x}\|^7 & s \le \|\boldsymbol{x}\| < s+2c\,,\\
\quad + D_{8}^{cc}\|\boldsymbol{x}\|^8 + D_{9}^{cc}\|\boldsymbol{x}\|^9 + D_{10}^{cc}\|\boldsymbol{x}\|^{10} + D_{11}^{cc}\|\boldsymbol{x}\|^{11}\,, & \\
0\,, & s+2c \le \|\boldsymbol{x}\|\,,
\end{cases}
$$

where

$$A_0^{cc} = \frac{231s^5-2772c^2s^3+7040c^3s^2-7128c^4s+2688c^5}{231s^5},$$
$$A_2^{cc} = \frac{-2\left(105s^3-432cs^2+630c^2s-320c^3\right)}{21s^5},$$
$$A_4^{cc} = \frac{3\left(56s^2-175cs+144c^2\right)}{35cs^5},$$
$$A_6^{cc} = \frac{-2\left(5s^2-12c^2\right)}{35c^3s^5},$$
$$A_7^{cc} = \frac{3}{56c^4s^3},$$
$$A_8^{cc} = \frac{-1}{42c^3s^5},$$
$$A_9^{cc} = \frac{-\left(s^2-6c^2\right)}{1680c^6s^5},$$
$$A_{11}^{cc} = \frac{-1}{36960c^6s^5},$$
$$B_{-1}^{cc} = \frac{-\left((s-2c)^8\right)\left(s^4+16cs^3+78c^2s^2+179c^3s+176c^4\right)}{36960c^6s^5},$$
$$B_0^{cc} = \frac{7s^{11}-330c^2s^9+1056c^3s^8+14784c^6s^5-177408c^8s^3+450560c^9s^2-456192c^{10}s+172032c^{11}}{29568c^6s^5},$$
$$B_1^{cc} = \frac{-\left((s-2c)^6\right)\left(s^4+12cs^3+54c^2s^2+148c^3s+216c^4\right)}{1120c^6s^5},$$
$$B_2^{cc} = \frac{5s^9-72c^2s^7-13440c^6s^3+55296c^7s^2-80640c^8s+40960c^9}{2688c^6s^5},$$
$$B_3^{cc} = \frac{-\left((s-2c)^4\right)\left(s^4+8cs^3+40c^2s^2+132c^3s+336c^4\right)}{448c^6s^5},$$
$$B_4^{cc} = \frac{3\left(s^7+14c^2s^5+1792c^5s^2-5600c^6s+4608c^7\right)}{2240c^6s^5},$$
$$B_5^{cc} = \frac{-\left((s-2c)^2\right)(s+4c)}{8c^3s^5},$$
$$B_6^{cc} = \frac{-(s+2c)\left(s^4-2cs^3+64c^2s^2+192c^3s-384c^4\right)}{2240c^6s^5},$$
$$B_7^{cc} = \frac{3(6s-5c)}{224c^4s^4},$$
$$B_8^{cc} = \frac{s^3-30c^2s-32c^3}{2688c^6s^5},$$
$$B_9^{cc} = \frac{-\left(s^2-6c^2\right)}{1120c^6s^5},$$
$$B_{10}^{cc} = \frac{1}{9856c^6s^4},$$
$$B_{11}^{cc} = \frac{-1}{24640c^6s^5},$$
$$C_{-1}^{cc} = \frac{-\left(s^{12}-66c^2s^{10}+275c^3s^9-1584c^5s^7+9504c^7s^5-70400c^9s^3-152064c^{10}s^2-134400c^{11}s-45056c^{12}\right)}{36960c^6s^5},$$
$$C_0^{cc} = \frac{7s^{11}-330c^2s^9+1056c^3s^8+14784c^6s^5-177408c^8s^3-450560c^9s^2-456192c^{10}s-172032c^{11}}{29568c^6s^5},$$
$$C_1^{cc} = \frac{-\left(s^{10}-30c^2s^8+60c^3s^7+336c^5s^5-8640c^7s^3-26880c^8s^2-32000c^9s-13824c^{10}\right)}{1120c^6s^5},$$
$$C_2^{cc} = \frac{5s^9-72c^2s^7-13440c^6s^3-55296c^7s^2-80640c^8s-40960c^9}{2688c^6s^5},$$
$$C_3^{cc} = \frac{-\left(s^8-28c^3s^5-672c^5s^3-4480c^6s^2-8640c^7s-5376c^8\right)}{448c^6s^5},$$
$$C_4^{cc} = \frac{3\left(s^7+14c^2s^5-1792c^5s^2-5600c^6s-4608c^7\right)}{2240c^6s^5},$$
$$C_5^{cc} = \frac{-(s-4c)(s+2c)^2}{8c^3s^5},$$

$$
\begin{aligned}
C_{6}^{cc} &= \frac{-(s-2c)\left(s^4+2cs^3+64c^2s^2-192c^3s-384c^4\right)}{2240c^6s^5},\\
C_{7}^{cc} &= \frac{3(2s-5c)}{224c^4s^4},\\
C_{8}^{cc} &= \frac{s^3-30c^2s+32c^3}{2688c^6s^5},\\
C_{9}^{cc} &= \frac{-\left(s^2-6c^2\right)}{3360c^6s^5},\\
C_{10}^{cc} &= \frac{1}{9856c^6s^4},\\
C_{11}^{cc} &= \frac{-1}{73920c^6s^5},\\
D_{-1}^{cc} &= \frac{(s+2c)^8\left(s^4-16cs^3+78c^2s^2-179c^3s+176c^4\right)}{36960c^6s^5},\\
D_{0}^{cc} &= \frac{-\left((s+2c)^7\right)\left(7s^4-98cs^3+454c^2s^2-1140c^3s+1344c^4\right)}{29568c^6s^5},\\
D_{1}^{cc} &= \frac{(s+2c)^6\left(s^4-12cs^3+54c^2s^2-148c^3s+216c^4\right)}{1120c^6s^5},\\
D_{2}^{cc} &= \frac{-\left((s+2c)^5\right)\left(5s^4-50cs^3+228c^2s^2-680c^3s+1280c^4\right)}{2688c^6s^5},\\
D_{3}^{cc} &= \frac{(s+2c)^4\left(s^4-8cs^3+40c^2s^2-132c^3s+336c^4\right)}{448c^6s^5},\\
D_{4}^{cc} &= \frac{-3(s+2c)^3\left(s^4-6cs^3+38c^2s^2-164c^3s+576c^4\right)}{2240c^6s^5},\\
D_{5}^{cc} &= \frac{-(s-4c)(s+2c)^2}{8c^3s^5},\\
D_{6}^{cc} &= \frac{(s+2c)\left(s^4-2cs^3+64c^2s^2+192c^3s-384c^4\right)}{2240c^6s^5},\\
D_{7}^{cc} &= \frac{-3(2s+5c)}{224c^4s^4},\\
D_{8}^{cc} &= \frac{-\left(s^3-30c^2s-32c^3\right)}{2688c^6s^5},\\
D_{9}^{cc} &= \frac{s^2-6c^2}{3360c^6s^5},\\
D_{10}^{cc} &= \frac{-1}{9856c^6s^4},\\
D_{11}^{cc} &= \frac{1}{73920c^6s^5}.
\end{aligned}
$$

Wendland $\Phi_{5,1}$ ball source

Again, we give formulas only for the $c > s$ case, and leave out the extremely large $\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_d$ formulas.

$$\begin{aligned}\Phi(\boldsymbol{x}) &= \left(1 - \frac{\|\boldsymbol{x}\|}{s}\right)_+^5 \left(5\frac{\|\boldsymbol{x}\|}{s} + 1\right), \quad \boldsymbol{x} \in \mathbb{R}^5. \\ &= \begin{cases} \left(1 - \frac{\|\boldsymbol{x}\|}{s}\right)^5 \left(5\frac{\|\boldsymbol{x}\|}{s} + 1\right), & 0 \le \|\boldsymbol{x}\| < s, \\ 0, & s \le \|\boldsymbol{x}\|. \end{cases}\end{aligned}$$

$$(\Phi \star_5 \mathcal{B}_c)(\boldsymbol{x}) =$$

$$\begin{cases} A_0^c + A_2^c\|\boldsymbol{x}\|^2 + A_4^c\|\boldsymbol{x}\|^4 + A_6^c\|\boldsymbol{x}\|^6 + A_8^c\|\boldsymbol{x}\|^8 \\ \quad + A_{10}^c\|\boldsymbol{x}\|^{10}, & 0 \le \|\boldsymbol{x}\| < c, \\ B_{-3}^c\|\boldsymbol{x}\|^{-3} + B_{-1}^c\|\boldsymbol{x}\|^{-1} + B_0^c + B_1^c\|\boldsymbol{x}\| + B_2^c\|\boldsymbol{x}\|^2 \\ \quad + B_3^c\|\boldsymbol{x}\|^3 + B_4^c\|\boldsymbol{x}\|^4 + B_5^c\|\boldsymbol{x}\|^5 + B_6^c\|\boldsymbol{x}\|^6, & c \le \|\boldsymbol{x}\| < s - c, \\ C_{-3}^c\|\boldsymbol{x}\|^{-3} + C_{-1}^c\|\boldsymbol{x}\|^{-1} + C_0^c + C_1^c\|\boldsymbol{x}\| + C_2^c\|\boldsymbol{x}\|^2 \\ \quad + C_3^c\|\boldsymbol{x}\|^3 + C_4^c\|\boldsymbol{x}\|^4 + C_5^c\|\boldsymbol{x}\|^5 + C_6^c\|\boldsymbol{x}\|^6 \\ \quad + C_7^c\|\boldsymbol{x}\|^7 + C_8^c\|\boldsymbol{x}\|^8 + C_9^c\|\boldsymbol{x}\|^9 + C_{10}^c\|\boldsymbol{x}\|^{10} & s - c \le \|\boldsymbol{x}\| < s + c, \\ \quad + C_{11}^c\|\boldsymbol{x}\|^{11}, \\ 0, & s + c \le \|\boldsymbol{x}\|, \end{cases}$$

where

$$\begin{aligned} A_0^c &= \tfrac{77s^6 - 825c^2s^4 + 1925c^3s^3 - 1925c^4s^2 + 924c^5s - 175c^6}{77s^6}, \\ A_2^c &= \tfrac{-15(s-c)^4}{s^6}, \\ A_4^c &= \tfrac{45(s-c)^2(2s-3c)}{7cs^6}, \\ A_6^c &= \tfrac{-5\left(4s^3 - 24c^2s + 21c^3\right)}{21c^3s^6}, \\ A_8^c &= \tfrac{5(s-2c)(s+2c)}{77c^5s^5}, \\ A_{10}^c &= \tfrac{12}{1001c^5s^5}, \\ B_{-3}^c &= \tfrac{-40c^6\left(13s^2 + 3c^2\right)}{3003s^5}, \\ B_{-1}^c &= \tfrac{40c^4\left(11s^2 + 4c^2\right)}{77s^5}, \\ B_0^c &= \tfrac{77s^6 - 825c^2s^4 - 1925c^4s^2 - 175c^6}{77s^6}, \\ B_1^c &= \tfrac{120c^2\left(3s^2 + 2c^2\right)}{7s^5}, \\ B_2^c &= \tfrac{-15\left(s^4 + 6c^2s^2 + c^4\right)}{s^6}, \end{aligned}$$

$$
\begin{aligned}
B_3^c &= \tfrac{40\left(7s^2+12c^2\right)}{7s^5},\\
B_4^c &= \tfrac{-45\left(7s^2+3c^2\right)}{7s^6},\\
B_5^c &= \tfrac{24}{s^5},\\
B_6^c &= \tfrac{-5}{s^6},\\
C_{-3}^c &= \tfrac{5(s+c)^9\left(9s^5-81cs^4+314c^2s^3-666c^3s^2+789c^4s-429c^5\right)}{768768c^5s^6},\\
C_{-1}^c &= \tfrac{-5(s+c)^7\left(7s^5-49cs^4+174c^2s^3-434c^3s^2+755c^4s-693c^5\right)}{19712c^5s^6},\\
C_0^c &= \tfrac{(s+c)^6\left(s^5-6cs^4+21c^2s^3-56c^3s^2+126c^4s-175c^5\right)}{154c^5s^6},\\
C_1^c &= \tfrac{-15(s+c)^5\left(s^5-5cs^4+18c^2s^3-50c^3s^2+157c^4s-441c^5\right)}{1792c^5s^6},\\
C_2^c &= \tfrac{-15(s+c)^4}{2s^6},\\
C_3^c &= \tfrac{5(s+c)^3\left(3s^5-9cs^4+46c^2s^3-114c^3s^2+1263c^4s+3675c^5\right)}{1792c^5s^6},\\
C_4^c &= \tfrac{-45(s+c)^2(2s+3c)}{14cs^6},\\
C_5^c &= \tfrac{-3(s+c)\left(s^5-cs^4+26c^2s^3-26c^3s^2-499c^4s-525c^5\right)}{256c^5s^6},\\
C_6^c &= \tfrac{5\left(4s^3-24c^2s-21c^3\right)}{42c^3s^6},\\
C_7^c &= \tfrac{45\left(s^4-14c^2s^2+21c^4\right)}{1792c^5s^6},\\
C_8^c &= \tfrac{-5(s-2c)(s+2c)}{154c^5s^5},\\
C_9^c &= \tfrac{5(s-c)(s+c)}{256c^5s^6},\\
C_{10}^c &= \tfrac{-6}{1001c^5s^5},\\
C_{11}^c &= \tfrac{15}{19712c^5s^6}.
\end{aligned}
$$

$$
\begin{aligned}
&(\Phi \star_5 \mathcal{B}_c \star_5 \mathcal{B}_c)(\boldsymbol{x}) = \\
&\begin{cases}
A_0^{cc} + A_2^{cc}\|\boldsymbol{x}\|^2 + A_4^{cc}\|\boldsymbol{x}\|^4 + A_6^{cc}\|\boldsymbol{x}\|^6 + A_8^{cc}\|\boldsymbol{x}\|^8 \\
\quad + A_9^{cc}\|\boldsymbol{x}\|^9 + A_{10}^{cc}\|\boldsymbol{x}\|^{10} + A_{11}^{cc}\|\boldsymbol{x}\|^{11} + A_{13}^{cc}\|\boldsymbol{x}\|^{13} & 0 \le \|\boldsymbol{x}\| < s - 2c\,, \\
\quad + A_{15}^{cc}\|\boldsymbol{x}\|^{15}\,, & \\
B_{-3}^{cc}\|\boldsymbol{x}\|^{-3} + B_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + B_0^{cc} + B_1^{cc}\|\boldsymbol{x}\| + B_2^{cc}\|\boldsymbol{x}\|^2 \\
\quad + B_3^{cc}\|\boldsymbol{x}\|^3 + B_4^{cc}\|\boldsymbol{x}\|^4 + B_5^{cc}\|\boldsymbol{x}\|^5 + B_6^{cc}\|\boldsymbol{x}\|^6 \\
\quad + B_7^{cc}\|\boldsymbol{x}\|^7 + B_8^{cc}\|\boldsymbol{x}\|^8 + B_9^{cc}\|\boldsymbol{x}\|^9 + B_{10}^{cc}\|\boldsymbol{x}\|^{10} & s - 2c \le \|\boldsymbol{x}\| < 2c\,, \\
\quad + B_{11}^{cc}\|\boldsymbol{x}\|^{11} + B_{12}^{cc}\|\boldsymbol{x}\|^{12} + B_{13}^{cc}\|\boldsymbol{x}\|^{13} + B_{14}^{cc}\|\boldsymbol{x}\|^{14} \\
\quad + B_{15}^{cc}\|\boldsymbol{x}\|^{15} + B_{16}^{cc}\|\boldsymbol{x}\|^{16}\,, & \\
C_{-3}^{cc}\|\boldsymbol{x}\|^{-3} + C_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + C_0^{cc} + C_1^{cc}\|\boldsymbol{x}\| + C_2^{cc}\|\boldsymbol{x}\|^2 \\
\quad + C_3^{cc}\|\boldsymbol{x}\|^3 + C_4^{cc}\|\boldsymbol{x}\|^4 + C_5^{cc}\|\boldsymbol{x}\|^5 + C_6^{cc}\|\boldsymbol{x}\|^6 \\
\quad + C_7^{cc}\|\boldsymbol{x}\|^7 + C_8^{cc}\|\boldsymbol{x}\|^8 + C_9^{cc}\|\boldsymbol{x}\|^9 + C_{10}^{cc}\|\boldsymbol{x}\|^{10} & 2c \le \|\boldsymbol{x}\| < s\,, \\
\quad + C_{11}^{cc}\|\boldsymbol{x}\|^{11} + C_{12}^{cc}\|\boldsymbol{x}\|^{12} + C_{13}^{cc}\|\boldsymbol{x}\|^{13} + C_{14}^{cc}\|\boldsymbol{x}\|^{14} \\
\quad + C_{15}^{cc}\|\boldsymbol{x}\|^{15} + C_{16}^{cc}\|\boldsymbol{x}\|^{16}\,, & \\
D_{-3}^{cc}\|\boldsymbol{x}\|^{-3} + D_{-1}^{cc}\|\boldsymbol{x}\|^{-1} + D_0^{cc} + D_1^{cc}\|\boldsymbol{x}\| \\
\quad + D_2^{cc}\|\boldsymbol{x}\|^2 + D_3^{cc}\|\boldsymbol{x}\|^3 + D_4^{cc}\|\boldsymbol{x}\|^4 + D_5^{cc}\|\boldsymbol{x}\|^5 \\
\quad + D_6^{cc}\|\boldsymbol{x}\|^6 + D_7^{cc}\|\boldsymbol{x}\|^7 + D_8^{cc}\|\boldsymbol{x}\|^8 + D_9^{cc}\|\boldsymbol{x}\|^9 & s \le \|\boldsymbol{x}\| < s + 2c\,, \\
\quad + D_{10}^{cc}\|\boldsymbol{x}\|^{10} + D_{11}^{cc}\|\boldsymbol{x}\|^{11} + D_{12}^{cc}\|\boldsymbol{x}\|^{12} + D_{13}^{cc}\|\boldsymbol{x}\|^{13} \\
\quad + D_{14}^{cc}\|\boldsymbol{x}\|^{14} + D_{15}^{cc}\|\boldsymbol{x}\|^{15} + D_{16}^{cc}\|\boldsymbol{x}\|^{16}\,, & \\
0\,, & s + 2c \le \|\boldsymbol{x}\|\,,
\end{cases}
\end{aligned}
$$

where

$$
\begin{aligned}
A_0^{cc} &= \tfrac{3003s^6 - 64350c^2s^4 + 224000c^3s^3 - 343200c^4s^2 + 258048c^5s - 78000c^6}{3003s^6}, \\
A_2^{cc} &= \tfrac{-5\left(3003s^4 - 16640cs^3 + 36036c^2s^2 - 35840c^3s + 13728c^4\right)}{1001s^6}, \\
A_4^{cc} &= \tfrac{15\left(352s^3 - 1617cs^2 + 2560c^2s - 1386c^3\right)}{539cs^6}, \\
A_6^{cc} &= \tfrac{-5\left(16s^3 - 128c^2s + 147c^3\right)}{147c^3s^6}, \\
A_8^{cc} &= \tfrac{5\left(7s^2 - 16c^2\right)}{539c^5s^5}, \\
A_9^{cc} &= \tfrac{-5}{336c^6s^3}, \\
A_{10}^{cc} &= \tfrac{12}{1001c^5s^5}, \\
A_{11}^{cc} &= \tfrac{5\left(s^2 - 8c^2\right)}{17248c^8s^5}, \\
A_{13}^{cc} &= \tfrac{-15\left(s^2 - 8c^2\right)}{3587584c^{10}s^5},
\end{aligned}
$$

$$A_{15}^{cc} = \frac{-1}{2690688c^{10}s^5},$$
$$B_{-3}^{cc} = \frac{-5(s-2c)^{12}\left(7s^7+168cs^6+1728c^2s^5+9440c^3s^4+31280c^4s^3+65292c^5s^2+81568c^6s+48048c^7\right)}{6952737792c^{10}s^6},$$
$$B_{-1}^{cc} = \frac{5(s-2c)^{10}\left(3s^7+60cs^6+524c^2s^5+2560c^3s^4+8208c^4s^3+18163c^5s^2+26300c^6s+19404c^7\right)}{30494464c^{10}s^6},$$
$$B_{0}^{cc} = \frac{-99s^{16}+3600c^2s^{14}-72800c^4s^{12}+159744c^5s^{11}+12300288c^{10}s^6}{24600576c^{10}s^6} + \frac{-263577600c^{12}s^4+917504000c^{13}s^3-1405747200c^{14}s^2+1056964608c^{15}s-319488000c^{16}}{24600576c^{10}s^6},$$
$$B_{1}^{cc} = \frac{15(s-2c)^{8}\left(2s^7+32cs^6+232c^2s^5+1024c^3s^4+3224c^4s^3+7575c^5s^2+12784c^6s+12348c^7\right)}{1793792c^{10}s^6},$$
$$B_{2}^{cc} = \frac{-45s^{14}+910c^2s^{12}-5720c^4s^{10}-7687680c^{10}s^4}{1025024c^{10}s^6} + \frac{42598400c^{11}s^3-92252160c^{12}s^2+91750400c^{13}s-35143680c^{14}}{1025024c^{10}s^6},$$
$$B_{3}^{cc} = \frac{5(s-2c)^{6}\left(4s^7+48cs^6+284c^2s^5+1168c^3s^4+3696c^4s^3+9389c^5s^2+18588c^6s+24500c^7\right)}{256256c^{10}s^6},$$
$$B_{4}^{cc} = \frac{-15\left(7s^{12}-44c^2s^{10}-264c^4s^8-360448c^9s^3+1655808c^{10}s^2-2621440c^{11}s+1419264c^{12}\right)}{1103872c^{10}s^6},$$
$$B_{5}^{cc} = \frac{3(s-2c)^{4}\left(s^7+8cs^6+40c^2s^5+160c^3s^4+560c^4s^3+1638c^5s^2+4144c^6s+7000c^7\right)}{39424c^{10}s^6},$$
$$B_{6}^{cc} = \frac{-5\left(s^{10}+6c^2s^8+112c^4s^6+8192c^7s^3-65536c^9s+75264c^{10}\right)}{150528c^{10}s^6},$$
$$B_{7}^{cc} = \frac{45(s-2c)^2(s+2c)^2}{1792c^5s^6},$$
$$B_{8}^{cc} = \frac{5(s+2c)\left(3s^6-6cs^5+68c^2s^4-136c^3s^3+3072c^4s^2+8192c^5s-16384c^6\right)}{2207744c^{10}s^5},$$
$$B_{9}^{cc} = \frac{-5\left(8s^3-7cs^2+4c^3\right)}{1792c^6s^6},$$
$$B_{10}^{cc} = \frac{-3\left(s^5+50c^2s^3-1400c^4s-2048c^5\right)}{1025024c^{10}s^5},$$
$$B_{11}^{cc} = \frac{15\left(4s^3-32c^2s+7c^3\right)}{137984c^8s^6},$$
$$B_{12}^{cc} = \frac{5\left(s^4-28c^2s^2+56c^4\right)}{2050048c^{10}s^6},$$
$$B_{13}^{cc} = \frac{-45\left(s^2-8c^2\right)}{7175168c^{10}s^5},$$
$$B_{14}^{cc} = \frac{15\left(s^2-2c^2\right)}{17425408c^{10}s^6},$$
$$B_{15}^{cc} = \frac{-1}{1793792c^{10}s^5},$$
$$B_{16}^{cc} = \frac{45}{2648662016c^{10}s^6},$$
$$C_{-3}^{cc} = \frac{-35s^{19}+2280c^2s^{17}-103360c^4s^{15}+406980c^5s^{14}-2351440c^7s^{12}+14780480c^9s^{10}-80620800c^{11}s^8}{6952737792c^{10}s^6} + \frac{405171200c^{13}s^6-2917232640c^{15}s^4-6879641600c^{16}s^3-7550484480c^{17}s^2-4233625600c^{18}s-984023040c^{19}}{6952737792c^{10}s^6},$$
$$C_{-1}^{cc} = \frac{15s^{17}-680c^2s^{15}+19040c^4s^{13}-54145c^5s^{12}+97240c^7s^{10}+583440c^9s^8}{30494464c^{10}s^6} + \frac{-9900800c^{11}s^6+133280000c^{13}s^4+398295040c^{14}s^3+537384960c^{15}s^2+362086400c^{16}s+99348480c^{17}}{30494464c^{10}s^6},$$
$$C_{0}^{cc} = \frac{-99s^{16}+3600c^2s^{14}-72800c^4s^{12}+159744c^5s^{11}+12300288c^{10}s^6}{24600576c^{10}s^6} + \frac{-263577600c^{12}s^4-917504000c^{13}s^3-1405747200c^{14}s^2-1056964608c^{15}s-319488000c^{16}}{24600576c^{10}s^6},$$
$$C_{1}^{cc} = \frac{30s^{15}-840c^2s^{13}+10920c^4s^{11}-15015c^5s^{10}-25740c^7s^8-480480c^9s^6}{1793792c^{10}s^6} + \frac{21840000c^{11}s^4+92252160c^{12}s^3+164640000c^{13}s^2+140574720c^{14}s+47416320c^{15}}{1793792c^{10}s^6},$$

$$C_2^{cc} = \frac{-45s^{14}+910c^2s^{12}-5720c^4s^{10}-7687680c^{10}s^4}{1025024c^{10}s^6} + \frac{-42598400c^{11}s^3-92252160c^{12}s^2-91750400c^{13}s-35143680c^{14}}{1025024c^{10}s^6},$$

$$C_3^{cc} = \frac{20s^{13}-260c^2s^{11}+2145c^5s^8+11440c^7s^6+572000c^9s^4}{256256c^{10}s^6} + \frac{5125120c^{10}s^3+14560000c^{11}s^2+17571840c^{12}s+7840000c^{13}}{256256c^{10}s^6},$$

$$C_4^{cc} = \frac{-15\left(7s^{12}-44c^2s^{10}-264c^4s^8+360448c^9s^3+1655808c^{10}s^2+2621440c^{11}s+1419264c^{12}\right)}{1103872c^{10}s^6},$$

$$C_5^{cc} = \frac{3\left(s^{11}-154c^5s^6-2200c^7s^4+61600c^9s^2+157696c^{10}s+112000c^{11}\right)}{39424c^{10}s^6},$$

$$C_6^{cc} = \frac{-5\left(s^{10}+6c^2s^8+112c^4s^6-8192c^7s^3+65536c^9s+75264c^{10}\right)}{150528c^{10}s^6},$$

$$C_7^{cc} = \frac{45(s-2c)^2(s+2c)^2}{1792c^5s^6},$$

$$C_8^{cc} = \frac{5(s-2c)\left(3s^6+6cs^5+68c^2s^4+136c^3s^3+3072c^4s^2-8192c^5s-16384c^6\right)}{2207744c^{10}s^5},$$

$$C_9^{cc} = \frac{-5\left(8s^3-21cs^2+12c^3\right)}{5376c^6s^6},$$

$$C_{10}^{cc} = \frac{-3\left(s^5+50c^2s^3-1400c^4s+2048c^5\right)}{1025024c^{10}s^5},$$

$$C_{11}^{cc} = \frac{5\left(4s^3-32c^2s+21c^3\right)}{137984c^8s^6},$$

$$C_{12}^{cc} = \frac{5\left(s^4-28c^2s^2+56c^4\right)}{2050048c^{10}s^6},$$

$$C_{13}^{cc} = \frac{-15\left(s^2-8c^2\right)}{7175168c^{10}s^5},$$

$$C_{14}^{cc} = \frac{15\left(s^2-2c^2\right)}{17425408c^{10}s^6},$$

$$C_{15}^{cc} = \frac{-1}{5381376c^{10}s^5},$$

$$C_{16}^{cc} = \frac{45}{2648662016c^{10}s^6},$$

$$D_{-3}^{cc} = \frac{5(s+2c)^{12}\left(7s^7-168cs^6+1728c^2s^5-9440c^3s^4+31280c^4s^3-65292c^5s^2+81568c^6s-48048c^7\right)}{6952737792c^{10}s^6},$$

$$D_{-1}^{cc} = \frac{-5(s+2c)^{10}\left(3s^7-60cs^6+524c^2s^5-2560c^3s^4+8208c^4s^3-18163c^5s^2+26300c^6s-19404c^7\right)}{30494464c^{10}s^6},$$

$$D_0^{cc} = \frac{(s+2c)^9\left(99s^7-1782cs^6+14220c^2s^5-65880c^3s^4+208880c^4s^3-475872c^5s^2+743616c^6s-624000c^7\right)}{24600576c^{10}s^6},$$

$$D_1^{cc} = \frac{-15(s+2c)^8\left(2s^7-32cs^6+232c^2s^5-1024c^3s^4+3224c^4s^3-7575c^5s^2+12784c^6s-12348c^7\right)}{1793792c^{10}s^6},$$

$$D_2^{cc} = \frac{5(s+2c)^7\left(9s^7-126cs^6+826c^2s^5-3500c^3s^4+11000c^4s^3-26768c^5s^2+48832c^6s-54912c^7\right)}{1025024c^{10}s^6},$$

$$D_3^{cc} = \frac{-5(s+2c)^6\left(4s^7-48cs^6+284c^2s^5-1168c^3s^4+3696c^4s^3-9389c^5s^2+18588c^6s-24500c^7\right)}{256256c^{10}s^6},$$

$$D_4^{cc} = \frac{15(s+2c)^5\left(7s^7-70cs^6+376c^2s^5-1520c^3s^4+4936c^4s^3-13264c^5s^2+28960c^6s-44352c^7\right)}{1103872c^{10}s^6},$$

$$D_5^{cc} = \frac{-3(s+2c)^4\left(s^7-8cs^6+40c^2s^5-160c^3s^4+560c^4s^3-1638c^5s^2+4144c^6s-7000c^7\right)}{39424c^{10}s^6},$$

$$D_6^{cc} = \frac{5(s+2c)^3\left(s^7-6cs^6+30c^2s^5-116c^3s^4+496c^4s^3-1824c^5s^2+5920c^6s-9408c^7\right)}{150528c^{10}s^6},$$

$$D_7^{cc} = \frac{45(s-2c)^2(s+2c)^2}{1792c^5s^6},$$

$$D_8^{cc} = \frac{-5(s+2c)\left(3s^6-6cs^5+68c^2s^4-136c^3s^3+3072c^4s^2+8192c^5s-16384c^6\right)}{2207744c^{10}s^5},$$

$$D_9^{cc} = \frac{5\left(8s^3+21cs^2-12c^3\right)}{5376c^6s^6},$$

$$D_{10}^{cc} = \frac{3\left(s^5+50c^2s^3-1400c^4s-2048c^5\right)}{1025024c^{10}s^5},$$
$$D_{11}^{cc} = \frac{-5\left(4s^3-32c^2s-21c^3\right)}{137984c^8s^6},$$
$$D_{12}^{cc} = \frac{-5\left(s^4-28c^2s^2+56c^4\right)}{2050048c^{10}s^6},$$
$$D_{13}^{cc} = \frac{15\left(s^2-8c^2\right)}{7175168c^{10}s^5},$$
$$D_{14}^{cc} = \frac{-15\left(s^2-2c^2\right)}{17425408c^{10}s^6},$$
$$D_{15}^{cc} = \frac{1}{5381376c^{10}s^5},$$
$$D_{16}^{cc} = \frac{-45}{2648662016c^{10}s^6}.$$

Bibliography

[1] Maxima, a Computer Algebra System (Version 5.16.2), 2008. `http://maxima.sourceforge.net/`.

[2] R.McN. Alexander. Elastic energy stores in running vertebrates. *American Zoologist*, 24(1):85–94, 1984.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, third edition, 1999.

[4] F.C. Anderson and M.G. Pandy. A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering*, 2(3):201–231, 1999.

[5] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artficial Intelligence Review*, 11:11–73, 1997.

[6] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning for control. *Artficial Intelligence Review*, 11:75–113, 1997.

[7] J. Auslander, A. Fukunaga, H. Partovi, J. Christensen, L. Hsu, P. Reiss, A. Shuman, J. Marks, and J.T. Ngo. Further experience with controller-based automatic motion synthesis for articulated figures. *ACM Transactions on Graphics*, 14(4):311–336, 1995.

[8] R.K. Beatson and H.-Q. Bui. Mollification formulas and implicit smoothing. Technical Report Research Report UCDMS 2003/19, University of Canterbury, 2003.

[9] R.K. Beatson and M.K. Langton. Integral interpolation. In A. Iske and J. Levesley, editors, *Algorithms for Approximation*, pages 199–218. Springer-Verlag, 2007.

[10] R.K. Beatson, W.A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740, 2000.

[11] L.L. Boneva, D. Kendall, and I. Stefanov. Spline transformations: Three new diagnostic aids for the statistical data-analyst. *Journal of the Royal Statistical Society, Series B*, 33:1–70, 1971.

[12] R. Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Sketches*, page 22. ACM, 2007.

[13] A.L. Brown. Uniform approximation by radial functions. In W. Light, editor, *Advances in Numerical Analysis II: Wavelets, Subdivision Algorithms and Radial Functions*, pages 203–206. Oxford University Press, 1992.

[14] I.E. Brown, E.J. Cheng, and G.E. Loeb. Measured and modeled properties of mammalian skeletal muscle. II. The effects of stimulus frequency on force-length and force-velocity relationships. *Journal of Muscle Research and Cell Motility*, 20:627–643, 1999.

[15] T. Buehrmann and E. Di Paolo. Biological actuators are not just springs: investigating muscle dynamics and control signals. In S. Nolfi et al., editors, *From Animals to Animats 9: 9th International Conference on Simulation of Adaptive Behavior (SAB'2006)*, pages 89–100, 2006.

[16] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of SIGGRAPH '01*, pages 67–76, 2001.

[17] G. Casciola, D. Lazzaro, L.B. Montefusco, and S. Morigi. Fast surface reconstruction and hole filling using positive definite radial basis functions. *Numerical Algorithms*, 39(1-3):289–305, 2005.

[18] Y.-L. Chen and S.-H. Lai. A partition-of-unity based algorithm for implicit surface reconstruction using belief propagation. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, pages 147–155. IEEE Computer Society, 2007.

[19] W. Cheney and W. Light. *A Course in Approximation Theory*. Brooks/Cole Publishing Company, 1999.

[20] J.-P. Chiles and P. Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. Wiley, 1999.

[21] J. Christensen, J. Marks, and J.T. Ngo. Automatic motion synthesis for 3D mass-spring models. Technical Report TR95-01, MERL, 1995.

[22] C. de Boor. Appendix to splines and histograms. In A. Sharma and A. Meir, editors, *Spline Functions and Approximation Theory*, volume 21 of *ISNM*, pages 329–358. Birkhauser-Verlag, 1973.

[23] J. Duchon. Splines minimizing rotation invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive theory of functions of several variables*, volume 571, pages 85–100. Springer, 1977.

[24] D. Dunbar and G. Humphreys. A spatial data structure for fast Poisson-disk sample generation. In *Proceedings of SIGGRAPH '06*, pages 503–508, 2006.

[25] N. Dyn and G. Wahba. On the estimation of functions of several variables from aggregated data. *Journal on Mathematical Analysis*, 13:134–152, 1982.

[26] P. Faloutsos. *Composable controllers for physics-based character animation*. PhD thesis, University of Toronto, 2002.

[27] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH '01*, pages 251–260. ACM Press, 2001.

[28] A.G. Feldman. Functional tuning of the nervous system with control of movements or maintenance of a steady posture: II. Controllable parameters of the muscle. *Biophysics*, 11:565–578, 1966.

[29] M.S. Floater and A. Iske. Multistep scattered data interpolation using compactly supported radial basis functions. *Journal of Computational and Applied Mathematics*, 73(1-2):65–78, 1996.

[30] C. Franke and R. Schaback. Solving partial differential equations by collocation using radial basis functions. *Applied Mathematics and Computation*, 93:73–82, 1998.

[31] R. Franke. Locally determined smooth interpolation at irregularly spaced points in several variables. *IMA Journal of Applied Mathematics*, 19(4):471–482, 1977.

[32] R. Franke. Smooth interpolation of scattered data by local thin plate splines. *Computers and Mathematics with Applications*, 8(4):273–281, 1982.

[33] R. Franke and G. Nielson. Smooth interpolation of large sets of scattered data. *International Journal for Numerical Methods in Engineering*, 15(11):1691–1704, 1980.

[34] A. Fukunaga, J. Marks, and J.T. Ngo. Automatic control of physically realistic animated figures using evolutionary programming. In *Third Annual Conference on Evolutionary Programming*, pages 76–83, 1994.

[35] B.A. Garner and M.G. Pandy. A kinematic model of the upper limb based on the visible human project. *Computer Methods in Biomechanics and Biomedical Engineering*, 2(2):107–124, 1999.

[36] B.A. Garner and M.G. Pandy. The obstacle-set method for representing muscle paths in musculoskeletal models. *Computer Methods in Biomechanics and Biomedical Engineering*, 3(1):1–30, 2000.

[37] B.A. Garner and M.G. Pandy. Musculoskeletal model of the upper limb based on the visible human male dataset. *Computer Methods in Biomechanics and Biomedical Engineering*, 4(2):93–126, 2001.

[38] B.A. Garner and M.G. Pandy. Estimation of musculotendon properties in the human upper limb. *Annals of Biomedical Engineering*, 31(2):207–220, 2003.

[39] L. Gritz and J.K. Hahn. Genetic programming for articulated figure motion. *The Journal of Visualization and Computer Animation*, 6(3):129–142, 1995.

[40] L. Gritz and J.K. Hahn. Genetic programming evolution of controllers for 3-D character animation. In J.R. Koza et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 139–146. Morgan Kaufmann, 1997.

[41] R. Grzeszczuk. *NeuroAnimator: Fast neural network emulation and control of physics-based models.* PhD thesis, University of Toronto, 1998.

[42] R. Grzeszczuk and D. Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH '95*, volume 29, pages 63–70, 1995.

[43] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. NeuroAnimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH '98*, pages 9–20, 1998.

[44] J.K. Hodgins and W.L. Wooten. Animating human athletes. In Y. Shirai and S. Hirose, editors, *Robotics Research: The Eighth International Symposium*, pages 356–367. Springer-Verlag, 1998.

[45] J.K. Hodgins, W.L. Wooten, D.C. Brogan, and J.F. O'Brien. Animating human athletics. In *Proceedings of SIGGRAPH '95*, pages 71–78, 1995.

[46] Y.C. Hon and R. Schaback. On unsymmetric collocation by radial basis functions. *Applied Mathematics and Computation*, 119(2-3):177–186, 2001.

[47] T.A. Hughes. *Measurement and Control Basics*. Instrument Society of America, 1988.

[48] P.A. Huijing. Length, shortening velocity, activation, and fatigue are not independent factors determining muscle force exerted. In J.M. Winters and P.E. Crago, editors, *Biomechanics and Neural Control of Posture and Movement*, chapter 5, pages 83–91. Springer-Verlag, 2000.

[49] Intel. Intel C++ Compiler Professional Edition for Linux (Version 10.1), 2008. `http://software.intel.com/en-us/intel-compilers/`.

[50] Intel. Intel Math Kernel Library (Version 10.0.4.023), 2008. `http://software.intel.com/en-us/intel-mkl/`.

[51] A. Iske. Reconstruction of functions from generalized Hermite-Birkhoff data. In C.K. Chui and L.L. Schumaker, editors, *Approximation Theory VIII, Vol 1, Approximation and Interpolation*, pages 257–264. World Scientific, 1995.

[52] A. Iske. Hierarchical scattered data filtering for multilevel interpolation schemes. In *Mathematical Methods for Curves and Surfaces: Oslo 2000*, pages 211–221. Vanderbilt University, 2001.

[53] A. Iske. *Multiresolution methods in scattered data modelling*, volume 37 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 2004.

[54] S. Jain, Y. Ye, and C.K. Liu. Optimization-based interactive motion synthesis. *ACM Transactions on Graphics*, 28(1), 2009.

[55] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python (Version 0.6.0), 2007. `http://www.scipy.org/`.

[56] E.J. Kansa. Multiquadrics—A scattered data approximation scheme with applications to computational fluid-dynamics—I: Surface approximations and partial derivative estimates. *Computers and Mathematics with Applications*, 19(8-9):127–145, 1990.

[57] E.J. Kansa. Multiquadrics—A scattered data approximation scheme with applications to computational fluid-dynamics—II: solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers and Mathematics with Applications*, 19(8-9):147–161, 1990.

[58] B.W. Kernighan and D.M. Ritchie. *The C programming language*. Prentice Hall, 2 edition, 1988.

[59] T. Komura, Y. Shinagawa, and T.L. Kunii. A muscle-based feed-forward controller of the human body. *Computer Graphics Forum*, 16(3):165–176, 1997.

[60] T. Komura, Y. Shinagawa, and T.L. Kunii. Creating and retargeting motion by the musculoskeletal human body model. *The Visual Computer*, 16(5):254–270, 2000.

[61] J. Laszlo. Controlling bipedal locomotion for computer animation. Master's thesis, University of Toronto, 1996.

[62] J. Laszlo, M. van de Panne, and E. Fiume. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of SIGGRAPH '96*, volume 30, pages 155–162, 1996.

[63] M.L. Latash. *Control of human movement*. Human Kinetics Publishers, 1993.

[64] M.L. Latash. *Neurophysiological basis of movement*. Human Kinetics Publishers, 2nd edition, 2008.

[65] D. Lazzaro and L.B. Montefusco. Radial basis functions for the multivariate interpolation of large scattered data sets. *Journal of Computational and Applied Mathematics*, 140(1-2):521–536, 2002.

[66] S.H. Lee and D. Terzopoulos. Heads up!: Biomechanical modeling and neuromuscular control of the neck. In *Proceedings of SIGGRAPH '06*, pages 1188–1198, 2006.

[67] W.A. Light. Some aspects of radial basis function approximation. In S.P. Singh, editor, *Approximation Theory, Spline Functions and Applications*, pages 163–190. Kluwer Academic, 1992.

[68] J. Lo, G. Huang, and D. Metaxas. Human motion planning based on recursive dynamics and optimal control techniques. *Multibody System Dynamics*, 8(4):433–458, 2002.

[69] G. Matheron. *The Theory of Regionalized Variables and its Applications*. École Nationale Supérieure des Mines de Paris, 1971.

[70] A.D. Maude. Interpolation—mainly for graph plotters. *Computer Journal*, 16(1):64–65, 1973.

[71] C.A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[72] A.W. Moore. *Efficient memory-based learning for robot control*. PhD thesis, University of Cambridge, 1990.

[73] A.W. Moore. Knowledge of knowledge and intelligent experimentation for learning control. In *Proceedings of the 1991 Seattle International Joint Conference on Neural Networks*, volume 2, pages 683–688, 1991.

[74] A.W. Moore. Fast, robust adaptive control by learning only forward models. In *Advances in Neural Information Processing Systems*, pages 571–578, 1992.

[75] A.W. Moore, C.G. Atkeson, and S.A. Schaal. Memory-based learning for control. Technical Report CMU-RI-TR-95-18, Carnegie Mellon University, 1995.

[76] B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the international conference on shape modeling & applications*, pages 89–98. IEEE Computer Society, 2001.

[77] F.J. Narcowich. Recent developments in approximation via positive definite functions. In C.K. Chui and L.L. Schumaker, editors, *Approximation Theory IX, Volume 2, Computational Aspects*, pages 221–242. Vanderbilt University Press, 1998.

[78] M. Neff and E. Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 81–88, 2002.

[79] J.T. Ngo and J. Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993.

[80] J.T. Ngo and J. Marks. Spacetime constraints revisited. In *Proceedings of SIGGRAPH '93*, pages 343–350, 1993.

[81] B.M. Nigg and W. Herzog, editors. *Biomechanics of the musculo-skeletal system*. John Wiley & Sons, 1994.

[82] P. Ning and J. Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41, 1993.

[83] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. In *Proceedings of SIGGRAPH '03*, pages 463–470. ACM, 2003.

[84] Y. Ohtake, A. Belyaev, and H.-P. Seidel. Multi-scale and adaptive CS-RBFs for shape reconstruction from cloud of points. In *MINGLE workshop on Multiresolution in Geometric Modelling*, pages 337–348, 2003.

[85] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003*, pages 153–161. IEEE Computer Society, 2003.

[86] Y. Ohtake, A. Belyaev, and H.-P. Seidel. 3D scattered data approximation with adaptive compactly supported radial basis functions. In *Shape Modeling Applications, 2004*, pages 31–39, 2004.

[87] Y. Ohtake, A. Belyaev, and H.-P. Seidel. 3D scattered data interpolation and approximation with multilevel compactly supported RBFs. *Graphical Models*, 67(3):150–165, 2005.

[88] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A composite approach to meshing scattered data. *Graphical Models*, 68(3):255–267, 2006.

[89] Y. Ohtake, A. Belyaev, and H.-P. Seidel. Sparse surface reconstruction with adaptive partition of unity and radial basis functions. *Graphical Models*, 68(1):15–24, 2006.

[90] M.G. Pandy. Computer modeling and simulation of human movement. *Annual Review of Biomedical Engineering*, 3:245–273, 2001.

[91] R. Pielot, M. Scholz, K. Obermayer, E.D. Gundelfinger, and A. Hess. A new approach to define landmarks for point-based warping in brain imaging. In *Bildverarbeitung fur die Medizin*, pages 28–32, 2000.

[92] R. Piessens, E. de Doncker-Kapenga, C.W. Uberhuber, and D.K. Kahaner. *QUADPACK: A subroutine package for automatic integration*. Springer-Verlag, 1983.

[93] J. Pouderoux, J.-C. Gonzato, I. Tobor, and P. Guitton. Adaptive hierarchical RBF interpolation for creating smooth digital elevation models. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 232–240. ACM, 2004.

[94] Paul S.A. Reitsma and N.S. Pollard. Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. In *Proceedings of SIGGRAPH '03*, pages 537–542, 2003.

[95] R.J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software*, 14(2):139–148, 1988.

[96] A. Ron and X. Sun. Strictly positive definite functions on spheres. *Mathematics of Computation*, 65(215):1513–1530, 1996.

[97] W. Rudin. *Real and complex analysis*. McGraw-Hill, 1987.

[98] D. Ruprecht and H. Müller. Free form deformation with scattered data interpolation methods. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling (Computing Suppl. 8)*, pages 267–281. Springer-Verlag, 1993.

[99] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[100] S. Schaal and C.G. Atkeson. Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14(1):57–71, 1994.

[101] R. Schaback. Convergence of unsymmetric kernel-based meshless collocation methods. *SIAM Journal on Numerical Analysis*, 45(1):333–351, 2007.

[102] R. Schaback and Z. Wu. Operators on radial functions. *Journal of Computational and Applied Mathematics*, 73:257–270, 1996.

[103] I.J. Schoenberg. Splines and histograms. In A. Sharma and A. Meir, editors, *Spline Functions and Approximation Theory*, volume 21 of *ISNM*, pages 277–327. Birkhauser-Verlag, 1973.

[104] R. Shadmehr. Equilibrium point hypothesis. In M.A. Arbib, S. Amari, and P.H. Arbib, editors, *The handbook of brain theory and neural networks*, pages 409–412. MIT Press, 2 edition, 2003.

[105] D. Shepard. A two dimensional interpolation function for irregular spaced data. In *Proceedings of the 23rd ACM National Conference*, pages 517–524, 1968.

[106] R. Smith. *Open dynamics engine user guide*, 2006. `http://www.ode.org`.

[107] R.L. Smith. *Intelligent motion control with an artificial cerebellum*. PhD thesis, University of Auckland, 1998.

[108] W.A. Stein et al. *Sage Mathematics Software (Version 3.1.2)*. The Sage Development Team, 2008. `http://www.sagemath.org`.

[109] B. Stroustrup. *The C++ programming language*. Addison-Wesley, special edition, 2000.

[110] X. Sun. Scattered Hermite interpolation using radial basis functions. *Linear Algebra and its Applications*, 207:135–146, 1994.

[111] SymPy Development Team. *SymPy: Python library for symbolic mathematics*, 2008. `http://www.sympy.org`.

[112] A.E. Tarwater. A parameter study of Hardy's multiquadric method for scattered data interpolation. Technical Report UCRL-54670, Lawrence Livermore National Laboratory, 1985.

[113] D.G. Thelen. Adjustment of muscle mechanics model parameters to simulate dynamic contractions in older adults. *Journal of Biomechanical Engineering*, 125(1):70–77, 2003.

[114] I. Tobor, P. Reuter, and C. Schlick. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. *Journal of WSCG*, 2004.

[115] I. Tobor, P. Reuter, and C. Schlick. Multi-scale reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Proceedings of Shape Modeling International 2004 (SMI'04)*, pages 19–30. IEEE Computer Society, 2004.

[116] I. Tobor, P. Reuter, and C. Schlick. Reconstructing multi-scale variational partition of unity implicit surfaces with attributes. *Graphical Models*, 68(1):25–41, 2006.

[117] N. Tronci, F. Molteni, and M. Bozzini. A comparison of local approximation methods for the analysis of meteorological data. *Archives for Meteorology, Geophysics, and Bioclimatology Series B*, 36(2):189–211, 1986.

[118] U.S. Geological Survey and National Geophysical Data Center. Digital aeromagnetic datasets for the conterminous United States and Hawaii—a companion to the North American Magnetic Anomaly Map. Technical Report OFR-02-361, United States Geological Survey, 2002. `http://pubs.usgs.gov/of/2002/ofr-02-361/`.

[119] M. van de Panne and E. Fiume. Sensor-actuator networks. In *Proceedings of SIGGRAPH '93*, pages 335–342, 1993.

[120] M. van de Panne, E. Fiume, and Z. Vranesic. Reusable motion synthesis using state-space controllers. In *Proceedings of SIGGRAPH '90*, pages 225–234, 1990.

[121] M. van de Panne, R. Kim, and E. Fiume. Synthesizing parameterized motions. In *Computer Animation and Simulation '94*, pages 1–14, 1994.

[122] M. van de Panne and A. Lamouret. Guided optimization for balanced locomotion. In D. Terzopoulos and D. Thalmann, editors, *Eurographics Workshop on Computer Animation and Simulation '95*, pages 165–177. Springer-Verlag, 1995.

[123] G. van Rossum et al. *Python Programming Language (Version 2.5.2)*. Python Software Foundation, 2008. `http://www.python.org/`.

[124] L.-Y. Wei. Parallel Poisson disk sampling. In *Proceedings of SIGGRAPH '08*, pages 1–9, 2008.

[125] R. Weinstein, E. Guendelman, and R. Fedkiw. Impulse-based PD control for joints and muscles. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 120, 2006.

[126] R.L. Weinstein. *Simulation and control of articulated rigid bodies*. PhD thesis, Stanford University, 2007.

[127] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995.

[128] H. Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. In C.K. Chui, L.L. Schumaker, and J. Stöckler, editors, *Approximation Theory X: Wavelets, Splines, and Applications*, pages 473–483. Vanderbilt University Press, 2002.

[129] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2005.

[130] A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of SIGGRAPH '88*, pages 159–168, 1988.

[131] J. Wu and L. Kobbelt. Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum*, 23(3):643–652, 2004.

[132] X. Wu, M.Y. Wang, and Q. Xia. Implicit fitting and smoothing using radial basis functions with partition of unity. In *CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 139–148. IEEE Computer Society, 2005.

[133] Z. Wu. Hermite-Birkhoff interpolation of scattered data by radial basis functions. *Approximation Theory and its Applications*, 8:1–10, 1992.

[134] Z. Wu. Multivariate compactly supported positive definite radial functions. *Advances in Computational Mathematics*, 4:283–292, 1995.

[135] Q. Xia, M.Y. Wang, and X. Wu. Orthogonal least squares in partition of unity surface reconstruction with radial basis function. In *Geometric Modeling and Imaging–New Trends 2006*, pages 28–33, 2006.

[136] H. Xie, J. Wang, J. Hua, H. Qin, and A. Kaufman. Piecewise C^1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 91–98. IEEE Computer Society, 2003.

[137] G.T. Yamaguchi, A.G.-U. Sawa, D.W. Moran, M.J. Fessler, and J.M. Winters. A survey of human musculotendon actuator properties. In J.M. Winters and S.L-Y. Woo, editors, *Multiple muscle systems: Biomechanics and movement organization*, chapter Appendix, pages 717–778. Springer-Verlag, 1990.

[138] K. Yin, K. Loken, and M. van de Panne. SIMBICON: Simple biped locomotion control. In *Proceedings of SIGGRAPH '07*, pages 105–114, 2007.

[139] F.E. Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical Reviews in Biomedical Engineering*, 17(4):359–411, 1989.